26st European Symposium on Algorithms

ESA 2018, August 20–22, 2018, Helsinki, Finland

Edited by

Yossi Azar
Hannah Bast
Grzegorz Herman

LIPICS

Editors

Yossi Azar	Hannah Bast	Grzegorz Herman
School of Computer Science	Department of Computer Science	Theoretical Computer Science
Tel Aviv University	University of Freiburg	Jagiellonian University in Kraków
`azar@tau.ac.il`	`bast@cs.uni-freiburg.de`	`gherman@tcs.uj.edu.pl`

ACM Classification 2012
Computer systems organization → Single instruction, multiple data;
Computing methodologies → Graphics processors; Robotic planning;
Hardware → Theorem proving and SAT solving;
Information systems → Data dictionaries;
Mathematics of computing → Approximation algorithms; Combinatorial algorithms; Combinatorial optimization; Combinatorics on words; Extremal graph theory; Graph algorithms; Graph theory; Network flows; Paths and connectivity problems; Permutations and combinations; Random graphs; Spectra of graphs;
Networks → Network design principles; Network structure;
Theory of computation → Algorithm design techniques; Algorithmic mechanism design; Approximation algorithms analysis; Cell probe models and lower bounds; Complexity theory and logic; Computational geometry; Database query processing and optimization (theory); Database theory; Data compression; Data structures and algorithms for data management; Data structures design and analysis; Design and analysis of algorithms; Distributed algorithms; Dynamic graph algorithms; Dynamic programming; Facility location and clustering; Fixed parameter tractability; Graph algorithms analysis; Integer programming; Linear programming; Market equilibria; Models of computation; Network games; Network optimization; Online algorithms; Oracles and decision trees; Packing and covering problems; Parallel algorithms; Parameterized complexity and exact algorithms; Probabilistic computation; Problems, reductions and completeness; Quantum computation theory; Routing and network design problems; Scheduling algorithms; Self-organization; Sorting and searching; Sparsification and spanners; Streaming models; Theory of randomized search heuristics

ISBN 978-3-95977-081-1

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-081-1.

Publication date
August, 2018

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

License
This work is licensed under a Creative Commons Attribution 3.0 Unported license (CC-BY 3.0): `http://creativecommons.org/licenses/by/3.0/legalcode`.
In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:
- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.ESA.2018.0
ISBN 978-3-95977-081-1 **ISSN 1868-8969** **http://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

Editorial Board

- Luca Aceto (*Chair*, Gran Sasso Science Institute and Reykjavik University)
- Susanne Albers (TU München)
- Christel Baier (TU Dresden)
- Javier Esparza (TU München)
- Michael Mitzenmacher (Harvard University)
- Madhavan Mukund (Chennai Mathematical Institute)
- Anca Muscholl (University Bordeaux)
- Catuscia Palamidessi (INRIA)
- Raimund Seidel (Saarland University and Schloss Dagstuhl – Leibniz-Zentrum für Informatik)
- Thomas Schwentick (TU Dortmund)
- Reinhard Wilhelm (Saarland University)

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Contents

Preface
Yossi Azar, Hannah Bast, and Grzegorz Herman 0:xi

Program Committees
.... 0:xiii

List of External Reviewers
.... 0:xv

Regular Papers

Algorithms for Inverse Optimization Problems
Sara Ahmadian, Umang Bhaskar, Laura Sanità, and Chaitanya Swamy 1:1–1:14

Two-Dimensional Maximal Repetitions
Amihood Amir, Gad M. Landau, Shoshana Marcus, and Dina Sokol 2:1–2:14

Approximate Convex Intersection Detection with Applications to Width and Minkowski Sums
Sunil Arya, Guilherme D. da Fonseca, and David M. Mount 3:1–3:14

On the Worst-Case Complexity of TimSort
Nicolas Auger, Vincent Jugé, Cyril Nicaud, and Carine Pivoteau 4:1–4:13

A New and Improved Algorithm for Online Bin Packing
János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin 5:1–5:14

Practical Access to Dynamic Programming on Tree Decompositions
Max Bannach and Sebastian Berndt 6:1–6:13

Average Whenever You Meet: Opportunistic Protocols for Community Detection
Luca Becchetti, Andrea Clementi, Pasin Manurangsi, Emanuele Natale, Francesco Pasquale, Prasad Raghavendra, and Luca Trevisan 7:1–7:13

Polynomial-Time Approximation Schemes for k-center, k-median, and Capacitated Vehicle Routing in Bounded Highway Dimension
Amariah Becker, Philip N. Klein, and David Saulpic 8:1–8:15

Fine-grained Lower Bounds on Cops and Robbers
Sebastian Brandt, Seth Pettie, and Jara Uitto 9:1–9:12

A Polynomial Kernel for Diamond-Free Editing
Yixin Cao, Ashutosh Rai, R. B. Sandeep, and Junjie Ye 10:1–10:13

Parallel and I/O-efficient Randomisation of Massive Networks using Global Curveball Trades
Corrie Jacobien Carstens, Michael Hamann, Ulrich Meyer, Manuel Penschuck, Hung Tran, and Dorothea Wagner 11:1–11:15

Space-Optimal Quasi-Gray Codes with Logarithmic Read Complexity
Diptarka Chakraborty, Debarati Das, Michal Koucký, and Nitin Saurabh 12:1–12:15

26th Annual European Symposium on Algorithms (ESA 2018).
Editors: Yossi Azar, Hannah Bast, and Grzegorz Herman
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A Framework for In-place Graph Algorithms
Sankardeep Chakraborty, Anish Mukherjee, Venkatesh Raman, and Srinivasa Rao Satti 13:1–13:16

Self-Assembly of Any Shape with Constant Tile Types using High Temperature
Cameron Chalk, Austin Luchsinger, Robert Schweller, and Tim Wylie 14:1–14:14

A Unified PTAS for Prize Collecting TSP and Steiner Tree Problem in Doubling Metrics
T-H. Hubert Chan, Haotian Jiang, and Shaofeng H.-C. Jiang 15:1–15:13

Near-Optimal Distance Emulator for Planar Graphs
Hsien-Chih Chang, Paweł Gawrychowski, Shay Mozes, and Oren Weimann 16:1–16:17

Approximation Schemes for Geometric Coverage Problems
Steven Chaplick, Minati De, Alexander Ravsky, and Joachim Spoerhase 17:1–17:15

Amortized Analysis of Asynchronous Price Dynamics
Yun Kuen Cheung and Richard Cole 18:1–18:15

Cycles to the Rescue! Novel Constraints to Compute Maximum Planar Subgraphs Fast
Markus Chimani and Tilo Wiedera 19:1–19:14

Parameterized Approximation Algorithms for Bidirected Steiner Network Problems
Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi 20:1–20:16

Online Facility Location with Deletions
Marek Cygan, Artur Czumaj, Marcin Mucha, and Piotr Sankowski 21:1–21:15

Improved Routing on the Delaunay Triangulation
Nicolas Bonichon, Prosenjit Bose, Jean-Lou De Carufel, Vincent Despré, Darryl Hill, and Michiel Smid 22:1–22:13

On Geometric Prototype and Applications
Hu Ding and Manni Liu 23:1–23:15

Improved Bounds for Multipass Pairing Heaps and Path-Balanced Binary Search Trees
Dani Dorfman, Haim Kaplan, László Kozma, Seth Pettie, and Uri Zwick 24:1–24:13

Improved Time and Space Bounds for Dynamic Range Mode
Hicham El-Zein, Meng He, J. Ian Munro, and Bryce Sandlund 25:1–25:13

Online Makespan Scheduling with Job Migration on Uniform Machines
Matthias Englert, David Mezlaf, and Matthias Westermann 26:1–26:14

Truthful Prompt Scheduling for Minimizing Sum of Completion Times
Alon Eden, Michal Feldman, Amos Fiat, and Tzahi Taub 27:1–27:14

Weighted Model Counting on the GPU by Exploiting Small Treewidth
Johannes K. Fichte, Markus Hecher, Stefan Woltran, and Markus Zisser 28:1–28:16

Light Spanners for High Dimensional Norms via Stochastic Decompositions
Arnold Filtser and Ofer Neiman 29:1–29:15

On the Tractability of Optimization Problems on H-Graphs
Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond 30:1–30:14

On the Optimality of Pseudo-polynomial Algorithms for Integer Programming
Fedor V. Fomin, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh 31:1–31:13

Symmetry Exploitation for Online Machine Covering with Bounded Migration
Waldo Gálvez, José A. Soto, and José Verschae 32:1–32:14

Edit Distance with Block Operations
Michał Gańczorz, Paweł Gawrychowski, Artur Jeż, and Tomasz Kociumaka 33:1–33:14

A QPTAS for Gapless MEC
Shilpa Garg and Tobias Mömke 34:1–34:14

FPT Algorithms for Embedding into Low Complexity Graphic Metrics
Arijit Ghosh, Sudeshna Kolay, and Gopinath Mishra 35:1–35:13

The Stochastic Score Classification Problem
Dimitrios Gkenosis, Nathaniel Grammel, Lisa Hellerstein, and Devorah Kletenik . 36:1–36:14

Improved Space-Time Tradeoffs for kSUM
Isaac Goldstein, Moshe Lewenstein, and Ely Porat 37:1–37:14

Dynamic Trees with Almost-Optimal Access Cost
Mordecai Golin, John Iacono, Stefan Langerman, J. Ian Munro, and Yakov Nekrich 38:1–38:14

A Tree Structure For Dynamic Facility Location
Gramoz Goranci, Monika Henzinger, and Dariusz Leniowski 39:1–39:13

Dynamic Effective Resistances and Approximate Schur Complement on Separable Graphs
Gramoz Goranci, Monika Henzinger, and Pan Peng 40:1–40:15

Buffered Count-Min Sketch on SSD: Theory and Experiments
Mayank Goswami, Dzejla Medjedovic, Emina Mekic, and Prashant Pandey 41:1–41:15

Scalable Katz Ranking Computation in Large Static and Dynamic Graphs
Alexander van der Grinten, Elisabetta Bergamini, Oded Green, David A. Bader, and Henning Meyerhenke 42:1–42:14

Round-Hashing for Data Storage: Distributed Servers and External-Memory Tables
Roberto Grossi and Luca Versari 43:1–43:14

Algorithmic Building Blocks for Asymmetric Memories
Yan Gu, Yihan Sun, and Guy E. Blelloch 44:1–44:15

On the Decision Tree Complexity of String Matching
Xiaoyu He, Neng Huang, and Xiaoming Sun 45:1–45:13

Decremental SPQR-trees for Planar Graphs
Jacob Holm, Giuseppe F. Italiano, Adam Karczmarz, Jakub Łącki, and Eva Rotenberg 46:1–46:16

Computing the Chromatic Number Using Graph Decompositions via Matrix Rank
Bart M. P. Jansen and Jesper Nederlof 47:1–47:15

Polynomial Kernels for Hitting Forbidden Minors under Structural Parameterizations
Bart M. P. Jansen and Astrid Pieterse 48:1–48:15

Quantum Algorithms for Connectivity and Related Problems
Michael Jarret, Stacey Jeffery, Shelby Kimmel, and Alvaro Piedrafita 49:1–49:13

Generalized Coloring of Permutations
Vít Jelínek, Michal Opler, and Pavel Valtr 50:1–50:14

Solving Partition Problems Almost Always Requires Pushing Many Vertices Around
Iyad Kanj, Christian Komusiewicz, Manuel Sorge, and Erik Jan van Leeuwen 51:1–51:14

String Attractors: Verification and Optimization
Dominik Kempa, Alberto Policriti, Nicola Prezza, and Eva Rotenberg 52:1–52:13

Data Reduction for Maximum Matching on Real-World Graphs: Theory and Experiments
Viatcheslav Korenwein, André Nichterlein, Rolf Niedermeier, and Philipp Zschoche 53:1–53:13

Searching a Tree with Permanently Noisy Advice
Lucas Boczkowski, Amos Korman, and Yoav Rodeh 54:1–54:13

Efficient and Adaptive Parameterized Algorithms on Modular Decompositions
Stefan Kratsch and Florian Nelles 55:1–55:15

On Nondeterministic Derandomization of Freivalds' Algorithm: Consequences, Avenues and Algorithmic Progress
Marvin Künnemann 56:1–56:16

Optimal Online Contention Resolution Schemes via Ex-Ante Prophet Inequalities
Euiwoong Lee and Sahil Singla 57:1–57:14

Equilibrium Computation in Atomic Splittable Routing Games
Umang Bhaskar and Phani Raj Lolakapuri 58:1–58:14

Online Non-Preemptive Scheduling to Minimize Weighted Flow-time on Unrelated Machines
Giorgio Lucarelli, Benjamin Moseley, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram 59:1–59:12

Finding Stable Matchings That Are Robust to Errors in the Input
Tung Mai and Vijay V. Vazirani 60:1–60:11

Disconnected Cuts in Claw-free Graphs
Barnaby Martin, Daniël Paulusma, and Erik Jan van Leeuwen 61:1–61:14

Practical Low-Dimensional Halfspace Range Space Sampling
Michael Matheny and Jeff M. Phillips 62:1–62:14

Nearly-Optimal Mergesorts: Fast, Practical Sorting Methods That Optimally Adapt to Existing Runs
J. Ian Munro and Sebastian Wild 63:1–63:16

On a Problem of Danzer
Nabil H. Mustafa and Saurabh Ray 64:1–64:8

Quasi-Polynomial Time Approximation Schemes for Packing and Covering Problems in Planar Graphs
Michał Pilipczuk, Erik Jan van Leeuwen, and Andreas Wiese 65:1–65:13

On Learning Linear Functions from Subset and Its Applications in Quantum Computing
Gábor Ivanyos, Anupam Prakash, and Miklos Santha 66:1–66:14

Strong Collapse for Persistence
Jean-Daniel Boissonnat, Siddharth Pritam, and Divyansh Pareek 67:1–67:13

On the Complexity of the (Approximate) Nearest Colored Node Problem
Maximilian Probst 68:1–68:14

Planar Support for Non-piercing Regions and Applications
Rajiv Raman and Saurabh Ray 69:1–69:14

An Exact Algorithm for the Steiner Forest Problem
Daniel R. Schmidt, Bernd Zey, and François Margot 70:1–70:14

Large Low-Diameter Graphs are Good Expanders
Michael Dinitz, Michael Schapira, and Gal Shahaf 71:1–71:15

Improved Dynamic Graph Coloring
Shay Solomon and Nicole Wein 72:1–72:16

Soft Subdivision Motion Planning for Complex Planar Robots
Bo Zhou, Yi-Jen Chiang, and Chee Yap 73:1–73:14

Preface

This volume contains the extended abstracts selected for presentation at ESA 2018, the 26th European Symposium on Algorithms, held in Helsinki, Finland, on 20–22 September 2018, as part of ALGO 2018. The scope of ESA includes original, high-quality, theoretical and applied research on algorithms and data structures. Since 2002, it has had two tracks: the Design and Analysis Track (Track A), intended for papers on the design and mathematical analysis of algorithms, and the Engineering and Applications Track (Track B), for submissions dealing with real-world applications, engineering, and experimental analysis of algorithms. Information on past symposia, including locations and proceedings, is maintained at http://esa-symposium.org.

In response to the call for papers for ESA 2018, 307 papers were submitted, 256 for Track A and 51 for Track B (these are the counts after the removal of papers with invalid format and after withdrawals). Paper selection was based on originality, technical quality, exposition quality, and relevance. Each paper received at least three reviews. The program committees selected 73 papers for inclusion in the program, 58 from Track A and 15 from Track B, yielding an acceptance rate of about 24%. In addition to the accepted contributions, the symposium featured two invited lectures: the first by Claire Mathieu (CNRS, Paris), and the second by Tim Roughgarden (Stanford University).

For this year's Track B, an experiment was performed, where the complete set of submissions was reviewed by two independent PCs. Each PC had 12 members, with a similar distribution according to gender, academic seniority, area of expertise, and continent of affiliation. In each PC, each submission was assigned to 3 PC members. Both PCs used the same standard reviewing process, which involved independently written reviews from the PC members, followed by an extensive discussion phase, and a voting phase for the papers that were still undecided in the end. Each PC eventually accepted 11 papers. A paper was accepted for Track B if and only if it was accepted by at least one of the two PCs. For the analysis of the process, the scores had a clearly communicated semantics and particular care was taken that for each submission in each PC the score set and the state of the discussion matched.

A detailed write-up of the course and the results of the experiment was still ongoing at the time of the creation of these proceedings. It will be published in a separate article containing the words "ESA 2018 experiment" in the title. As an appetizer, here is a list of some of the questions investigated and a first informal answer: how large was the overlap of the set of accepted papers by the two PCs (it fluctuated between 50% and 75% throughout the reviewing process and was very sensitive to relatively minor changes in the discussion), how many "clear accepts" were there (none really: the chance that a paper with the high score in one PC also had the high score in the other PC was not larger than random), how many "clear rejects" were there (about one fourth of all submissions had only negative reviews in both PCs, and the overlap of these sets from the two PCs was over 70%), how many papers had overall positive reviews in one PC and overall negative reviews in the other PC (less than 10% of all submissions), how effective were the discussion phase and the final voting phase (it's not clear that either had a non-random effect on the set of papers that were eventually accepted), what are possible implications for future PCs (read the publication when it's there).

The European Association for Theoretical Computer Science (EATCS) sponsored a best paper award and a best student paper award. A submission was eligible for the best student paper award if all authors were doctoral, master, or bachelor students at the time of submission. The best student paper award for Track A was given to Maximilian Probst for the paper “On the complexity of the (approximate) nearest colored node problem”. The best student paper award for Track B was given to Max Bannach and Sebastian Berndt for the paper “Practical Access to Dynamic Programming on Tree Decompositions”. The best paper award for Track A was given to Jacob Holm, Giuseppe F. Italiano, Adam Karczmarz, Jakub Łącki and Eva Rotenberg for the paper “Decremental SPQR-trees for Planar Graphs”. The best paper award for Track B was given to Daniel R. Schmidt, Bernd Zey and François Margot for the paper “An exact algorithm for the Steiner forest problem”.

We wish to thank all the authors who submitted papers for consideration, the invited speakers, the members of the program committees for their hard work, and all the external reviewers who assisted the program committees in the evaluation process. Special thanks go to the local organizing committee, who helped us with the organization of the conference.

Program Committees

Track A (Design and Analysis) Program Committee

- Yossi Azar *(chair)* (Tel Aviv University, Israel)
- Petra Berenbrink (University of Hamburg, Germany)
- Shuchi Chawla (University of Wisconsin-Madison, USA)
- Flavio Chierichetti (Sapienza University of Rome, Italy)
- Ashish Chiplunkar (EPFL, Switzerland)
- George Christodoulou (University of Liverpool, UK)
- Samuel Fiorini (Université libre de Bruxelles, Belgium)
- Cyril Gavoille (University of Bordeaux, France)
- Loukas Georgiadis (University of Ioannina, Greece)
- Anupam Gupta (Carnegie Mellon University, USA)
- Danny Hermelin (Ben Gurion University, Israel)
- Zhiyi Huang (University of Hong Kong, China)
- Satoru Iwata (University of Tokyo, Japan)
- Klaus Jansen (University of Kiel, Germany)
- Thomas Kesselheim (TU Dortmund, Germany)
- Lukasz Kowalik (University of Warsaw, Poland)
- Sebastian Krinninger (University of Salzburg, Austria)
- Amit Kumar (IIT Delhi, India)
- Daniel Lokshtanov (University of Bergen, Norway)
- Konstantin Makarychev (Northwestern University, USA)
- Debmalya Panigrahi (Duke University, USA)
- Merav Parter (Weizmann Institute, Israel)
- Christian Scheideler (Paderborn University, Germany)
- Bettina Speckmann (TU Eindhoven, Netherlands)
- Subhash Suri (University of California Santa Barbara, USA)
- Csaba D. Tóth (Cal State Northridge, USA)
- Gerhard Woeginger (RWTH Aachen, Germany)
- Christian Wulff-Nilsen (University of Copenhagen, Denmark)

Track B (Engineering and Applications) Program Committee

- Martin Aumüller (IT University of Copenhagen, Denmark)
- Hannah Bast *(chair)* (University of Freiburg, Germany)
- Christina Büsing (RWTH Aachen University, Germany)
- Pierluigi Crescenzi (University of Florence, Italy)
- Veronica Gil-Costa (Universidad Nacional de San Luis, Argentina)
- Michael T. Goodrich (University of California, Irvine, USA)
- Paolo Ferragina (University of Pisa, Italy)
- Stefan Funke (University of Stuttgart, Germany)
- Inge Li Gørtz (Technical University of Denmark, Denmark)
- Sungjin Im (University of California at Merced, USA)
- Michael Kerber (Graz University of Technology, Austria)
- Silvio Lattanzi (Google, Switzerland)
- Jon Lee (University of Michigan, USA)
- Tamara Mchedlidze (Karlsruhe Institute of Technology, Germany)
- Matthias Müller-Hannemann (Martin Luther University Halle-Wittenberg, Germany)
- Petra Mutzel (TU Dortmund University, Germany)
- Gonzalo Navarro (University of Chile, Chile)
- Richard Peng (Georgia Institute of Technology, USA)
- Simon J. Puglisi (University of Helsinki, Finland)
- Melanie Schmidt (University of Bonn, Germany)
- Anita Schöbel (Georg-August-Universität Göttingen, Germany)
- Chris Schwiegelshohn (Sapienza University of Rome, Italy)
- Sebastian Stiller (TU Braunschweig, Germany)
- Darren Strash (Colgate University, USA)
- Carola Wenk (Tulane University, USA)

List of External Reviewers

Amir Abboud
Mikkel Abrahamsen
Jayadev Acharya
Peyman Afshani
Akanksha Agrawal
Oswin Aichholzer
Yaroslav Akhremtsev
Hugo Akitaya
Eleni C. Akrida
Gorjan Alagic
Xavier Allamigeon
Noga Alon
Helmut Alt
Amihood Amir
Haris Angelidakis
Antonios Antoniadis
Srinivasan Arunachalam
James Aspnes
Igor Averbakh
Davide Bacciu
Arturs Backurs
Eric Balkanski
Evangelos Bampas
Evripidis Bampis
Hideo Bannai
Nikhil Bansal
Jérémy Barbay
Lukas Barth
Ulrich Bauer
Ruben Becker
Xiaohui Bei
Djamal Belazzougui
Mark de Berg
Sebastian Berndt
Sayan Bhattacharya
Arnab Bhattacharyya
Marcin Bienkowski
Gianfranco Bilardi
Philip Bille
Vittorio Bilò
Timo Bingmann
Andreas Björklund
Thomas Bläsius
Hans L. Bodlaender
Greg Bodwin
Maria Luisa Bonet
Édouard Bonnet
Steffen Borgwardt
Piotr Borowiecki
Ulrik Brandes
Marco Bressan
Valentin Brimkov
Karl Bringmann
Simina Brânzei
Gerandy Brito
Niv Buchbinder
Mickaël Buchet
Valentin Buchhold
Kevin Buchin
Maike Buchin
Boris Bukh
Christina Burt
Sebastian Buschjäger
Sam Buss
Matthias Buttkus
Jaroslaw Byrka
Sergio Cabello
Manuel Cáceres
Chris Cade
Yixin Cao
Tim Carpenter
Matteo Ceccarello
Parinya Chalermsook
Erin Chambers
T-H. Hubert Chan
Timothy M. Chan
Karthekeyan Chandrasekaran
Panagiotis Charalampopoulos
Chandra Chekuri
Jiehua Chen
Lin Chen
Yu Cheng
Rajesh Chitnis
Janka Chlebikova
Anamitra Roy Choudhury
Tobias Christiani
Anders Roy Christiansen
Timothy Chu
Will Cipolli
Catherine Cleophas

Dustin Cobas
Avi Cohen
Alessio Conte
Graham Cormode
Denis Cornaz
Bruno Courcelle
Alex Cozzi
Ágnes Cseh
Radu Curticapean
Artur Czumaj
Christine Dahn
Abhimanyu Das
Syamantak Das
Sanjeeb Dash
Sina Dehghani
Holger Dell
Max Deppert
Nikhil Devanur
Tamal Dey
Martin Dietzfelbinger
Michael Dinitz
Paul Dorbec
David Doty
Agostino Dovier
Feodor Dragan
Matt Drescher
Anne Driemel
Andre Droschinsky
Ran Duan
Philippe Duchon
Paul Duetting
Stephane Durocher
Martijn van Ee
Eduard Eiben
Khaled Elbassioni
Alina Ene
David Eppstein
Thomas Erlebach
Elaine Eschen
Louis Esperet
Mikko Berggren Ettienne
Rolf Fagerberg
Brittany Fasy
John Fearnley
Uriel Feige
Moran Feldman
Andreas Emil Feldmann
Michael Feldmann
Henning Fernau
Diodato Ferraioli
Amos Fiat
Hendrik Fichtenberger
Gabriele Fici
Arnold Filtser
Anja Fischer
Felix Fischer
Matthias Fischer
Till Fluschnik
Dimitris Fotakis
Kyle Fox
Tom Friedetzky
Tobias Friedrich
Alan Frieze
Zachary Friggstad
Ulderico Fugacci
Toshihiro Fujito
Ben Fulcher
Radoslav Fulek
Travis Gagie
Waldo Gálvez
Guilhem Gamard
Arun Ganesh
Arnab Ganguly
Wilfried Gansterer
Naveen Garg
Bernd Gärtner
Paweł Gawrychowski
Rong Ge
Ofir Geri
Yiannis Giannakopoulos
Konstantinos Giannis
Panos Giannopoulos
Archontia Giannopoulou
Vasilis Gkatzelis
Alexander Göke
Shay Golan
Kira Goldner
Petr Golovach
Adrián Gómez-Brandón
Gramoz Goranci
Thorsten Götte
Lee-Ad Gottlieb
Garance Gourdel
Vineet Goyal
Daniel Graf
Fabrizio Grandoni

Nick Gravin
Elena Grigorescu
Martin Gronemann
Martin Groß
Roberto Grossi
Krystal Guo
Manoj Gupta
Shahrzad Haddadan
Michael Hamann
Samuel Haney
Kristoffer Arnsfelt Hansen
Thomas Dueholm Hansen
Nicolas Hanusse
Tobias Harks
Hamed Hatami
Elham Havvaei
Michael Hay
Meng He
Christoph Helmberg
Monika Henzinger
John Hershberger
Mhand Hifi
Kristian Hinnenthal
Martin Hoefer
Michael Hoffmann
Jacob Holm
Ivor Hoog V.D.
Chien-Chung Huang
Patricio Huepe
Thore Husfeldt
John Iacono
Alonso Inostrosa-Psijas
Takehiro Ito
Yoichi Iwata
Taisuke Izumi
Adalat Jabrayilov
Riko Jacob
Lars Jaffke
Ragesh Jaiswal
Bart M. P. Jansen
Bruno Jartoux
Artur Jeż
Łukasz Jeż
Shaofeng Jiang
Kai Jin
Timothy Johnson
Jordan Jorgensen
Alpár Jüttner
Dominik Kaaser
Volker Kaibel
Naonori Kakimura
Christos Kalaitzis
Sagar Kale
Naoyuki Kamiyama
Frank Kammer
Haim Kaplan
Michael Kapralov
Aikaterini Karanasiou
Andreas Karrenbauer
Michael Kaufmann
Bart de Keijzer
Nathaniel Kell
Jonathan Kelner
Dominik Kempa
Balázs Keszegh
Arindam Khan
Shuji Kijima
Eun Jung Kim
Ralf Klasing
Jonathan Klawitter
Kim-Manuel Klein
Philip Klein
Lasse Kliemann
Max Klimm
Peter Kling
Yusuke Kobayashi
Tomasz Kociumaka
Zhuan Khye Koh
Sudeshna Kolay
Christina Kolb
Christian Komusiewicz
Spyros Kontogiannis
Parisa Kordjamshidi
Janne H. Korhonen
Arie Koster
Irina Kostitsyna
Martin Koutecký
Ioannis Koutis
Laszlo Kozma
Rastislav Kralovic
Dieter Kratsch
Stefan Kratsch
Marc Van Kreveld
Ravishankar Krishnaswamy
R. Krithika
Michael Krivelevich

Amer Krivošija
Sven Krumke
Dominik Krupke
Piotr Krysta
Janardhan Kulkarni
Neeraj Kumar
Niraj Kumar
Marvin Künnemann
Anastasia Kurdia
Denis Kurz
Anthony Labarre
Sébastien Labbé
Bundit Laekhanukit
Michael Lampis
Stefan Langerman
Alexandra Lassota
Luigi Laura
Francois Le Gall
Euiwoong Lee
Christoph Lenzen
Stefano Leucci
Roie Levin
Jason Li
Shi Li
Andre Lieutier
Nutan Limaye
Andre Linhares
Jinyan Liu
Elisabeth Lobe
Maarten Löffler
Veronika Loitzenbauer
Marten Maack
Sepideh Mahabadi
Yury Makarychev
Frederik Mallmann-Trenn
Florin Manea
Sebastian Maneth
George Manoussakis
Pasin Manurangsi
Giovanni Manzini
Jieming Mao
Andrea Marino
Samuel McCauley
Andrew McGregor
Nicole Megow
Aranyak Mehta
Julian Mestre
Wouter Meulemans
Ulrich Meyer
Othon Michail
Samuel Micka
Ivan Mikhailin
Benjamin Miller
David L. Millman
Till Miltzow
Majid Mirzanezhad
Pranabendu Misra
Slobodan Mitrovic
Matthias Mnich
Hendrik Molter
Christopher Morris
Benjamin Moseley
Michal Moshkovitz
David Mount
Aidasadat Mousavifar
Wolfgang Mulzer
Ralf-Peter Mundani
Cameron Musco
Viswanath Nagarajan
Meghana Nasre
Amir Nayyeri
Jesper Nederlof
Ofer Neiman
Hung Nguyen
Rad Niazadeh
André Nichterlein
Rolf Niedermeier
Alexander Noe
Christos Nomikos
Ashkan Norouzi Fard
Zeev Nutov
Pascal Ochem
Carlos Ochoa
Lutz Oettershagen
Eunjin Oh
Yoshio Okamoto
Aurélien Ooms
Tim Ophelders
Sebastian Ordyniak
Joseph O'Rourke
Pekka Orponen
Rotem Oshman
Sang-Il Oum
Maris Ozols
Linda Pagli
Katarzyna Paluch

Alessandro Panconesi
Fahad Panolan
Greta Panova
Charis Papadopoulos
Kunsoo Park
Nikos Parotsidis
Maurizio Patrignani
Christophe Paul
Niklas Paulsen
Ami Paz
Pan Peng
Richard Peng
Pablo Pérez-Lantero
Ljubomir Perkovic
Giulio Ermanno Pibiri
Marcin Pilipczuk
Michał Pilipczuk
Ely Porat
Giuseppe Prencipe
Nicola Prezza
Maximilian Probst
Ioannis Psarros
Simon Puglisi
Kent Quanrud
Marcel Radermacher
Sharath Raghvendra
Benjamin Raichel
M. S. Ramanujan
Michael Raskin
Malin Rau
Jean-Florent Raymond
Ilya Razenshteyn
Igor Razgon
David Renault
David Richerby
Havana Rika
Matteo Riondato
Lars Rohwedder
Clemens Rösner
Günter Rote
Eva Rotenberg
Alan Roytman
Paweł Rzążewski
Yogish Sabharwal
Kunihiko Sadakane
Barna Saha
Rahul Saladi
Piotr Sankowski
Srinivasa Rao Satti
Maria Saumell
Saket Saurabh
Till Schäfer
Oliver Schaudt
Christian Scheffer
Kevin Schewior
Ingo Schiermeyer
Christian Schindelhauer
Sebastian Schlag
Andreas Schmid
Arne Schmidt
Daniel Schmidt
Melanie Schmidt
Jon Schneider
Christian Schulz
Jordan Schupbach
Gregory Schwartzman
Pascal Schweitzer
Chris Schwiegelshohn
Uwe Schwiegelshohn
Diego Seco
Saeed Seddighin
Víctor Sepúlveda
Alexander Setzer
Alkmini Sgouritsa
Mordechai Shalom
Roohani Sharma
Nobutaka Shimizu
Akiyoshi Shioura
Julian Shun
Aaron Sidford
Anastasios Sidiropoulos
Francesco Silvestri
Sahil Singla
Stavros Sintos
Carsten Sinz
Jouni Sirén
Nodari Sitchinava
Primoz Skraba
Martin Skutella
Shakhar Smorodinsky
Roberto Solar
Frank Sommer
Anthony Man-Cho So
Christian Sohler
Shay Solomon
Kiril Solovey

Manuel Sorge
M. Grazia Speranza
Frits Spieksma
Sophie Spirkl
Frank Staals
Georgios Stamoulis
Rob van Stee
Mike Steel
Ben Strasser
Peter J. Stuckey
Hsin-Hao Su
Torsten Suel
Warut Suksompong
He Sun
Maxim Sviridenko
Meesum Syed Mohammad
Zhihao Gavin Tang
Shin-Ichi Tanigawa
Gábor Tardos
Kavitha Telikepalli
Yifeng Teng
Veerle Timmermans
Sumedh Tirodkar
Andreas Tönnis
Ohad Trabelsi
Guillermo Trabes
Nicolas Trotignon
Kostas Tsichlas
Charalampos Tsourakakis
Dekel Tsur
Torsten Ueckerdt
Marc Uetz
Chris Umans
Ali Vakilian
Erik Jan van Leeuwen
Kasturi Varadarajan
Shai Vardi
Vincent Vatter
Kevin Verbeek
Luca Versari
Laurent Viennot
Aravindan Vijayaraghavan
Fabio Vitale
Ben Lee Volk
Hoa Vu
Magnus Wahlström
Tomasz Waleń
Haitao Wang
Jianxin Wang
Yuyi Wang
Justin Ward
Karol Węgrzycki
Karsten Weihe
Oren Weimann
S. Matthew Weinberg
Armin Weiss
Stefan Weltge
Andreas Wiese
Virginia Williams
Carsten Witt
Damien Woods
Marcin Wrochna
Xiaowei Wu
Chao Xu
Pan Xu
Yutaro Yamaguchi
Chunxing Yin
Eylon Yogev
Huacheng Yu
Xilin Yu
Luca Zanetti
Meirav Zehavi
Wei Zhan
Guochuan Zhang
Qin Zhang
Yuhao Zhang
Samson Zhou
Yuan Zhou
Xue Zhu
Uri Zwick
Anna Zych-Pawlewicz

Algorithms for Inverse Optimization Problems

Sara Ahmadian
Combinatorics and Optimization, Univ. Waterloo, Waterloo, ON N2L 3G1, Canada
sahmadian@uwaterloo.ca

Umang Bhaskar[1]
Tata Institute of Fundamental Research, Mumbai, India 400 005
umang@tifr.res.in

Laura Sanità
Combinatorics and Optimization, Univ. Waterloo, Waterloo, ON N2L 3G1, Canada
sanita@uwaterloo.ca

Chaitanya Swamy[2]
Combinatorics and Optimization, Univ. Waterloo, Waterloo, ON N2L 3G1, Canada
cswamy@uwaterloo.ca

Abstract

We study *inverse optimization problems*, wherein the goal is to map *given solutions* to an underlying optimization problem to a *cost vector* for which the given solutions are the (unique) optimal solutions. Inverse optimization problems find diverse applications and have been widely studied. A prominent problem in this field is the *inverse shortest path* (ISP) problem [9, 3, 4], which finds applications in shortest-path routing protocols used in telecommunications. Here we seek a cost vector that is positive, *integral*, induces a set of given paths as the unique shortest paths, and has minimum ℓ_∞ norm. Despite being extensively studied, very few algorithmic results are known for inverse optimization problems involving integrality constraints on the desired cost vector whose norm has to be minimized.

Motivated by ISP, we initiate a systematic study of such integral inverse optimization problems from the perspective of designing polynomial time approximation algorithms. For ISP, our main result is an *additive* 1-approximation algorithm for multicommodity ISP with node-disjoint commodities, which we show is *tight* assuming $P \neq NP$. We then consider the integral-cost inverse versions of various other fundamental combinatorial optimization problems, including min-cost flow, max/min-cost bipartite matching, and max/min-cost basis in a matroid, and obtain tight or nearly-tight approximation guarantees for these. Our guarantees for the first two problems are based on results for a broad generalization, namely *integral inverse polyhedral optimization*, for which we also give approximation guarantees. Our techniques also give similar results for variants, including ℓ_p-norm minimization of the integral cost vector, and distance-minimization from an initial cost vector.

2012 ACM Subject Classification Theory of computation → Network optimization, Theory of computation → Approximation algorithms analysis, Mathematics of computing → Network flows

Keywords and phrases Inverse optimization, Shortest paths, Approximation algorithms, Linear programming, Polyhedral theory, Combinatorial optimization

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.1

[1] Funded in part by a Ramanujan Fellowship. Part of this work was done while visiting U. Waterloo.
[2] Supported in part by NSERC grant 327620-09 and an NSERC Discovery Accelerator Supplement Award.

© Sara Ahmadian, Umang Bhaskar, Laura Sanità, and Chaitanya Swamy;
licensed under Creative Commons License CC-BY
26th Annual European Symposium on Algorithms (ESA 2018).
Editors: Yossi Azar, Hannah Bast, and Grzegorz Herman; Article No. 1; pp. 1:1–1:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Consider the following problem, adapted from [4], faced by the administrator of a telecommunication network. The administrator seeks to impose a desired routing pattern (e.g., one that distributes traffic along multiple paths to minimize congestion) under a given underlying routing protocol. Many routing protocols – OSPF (open-shortest-path-first), IS-IS etc. – use shortest-path routing, with path lengths defined as the sum of link lengths that are set by the administrator, where the link lengths must typically be positive integers that can be stored using a limited number of bits (e.g., in IS-IS, they must be at most 63). Thus, the administrator must choose small, positive, integer link lengths so that the resulting shortest paths *coincide* with the prescribed paths (thus ensuring that we utilize precisely these paths).

This is an *inverse shortest path* (ISP) problem (which also arises in seismic tomography, traffic modeling, and network tolling [9, 19, 8, 12, 13]), a prominent problem from the rich class of *inverse optimization problems*, wherein we are *given solutions* to an underlying optimization problem and we *seek a cost vector* under which the given solutions constitute the (unique) optimal solutions. Since we map solutions to a suitable cost vector, this is termed inverse optimization. Inverse optimization problems find applications in a variety of domains including telecommunication routing [3, 4], seismic and medical tomography [9, 19, 23], traffic modeling and network tolling [12, 13, 9, 8], and portfolio optimization [18]. They also arise in the domain of *revealed-preference theory* in economics [24], which seeks to understand when observations can be attributed to behavior consistent with game-theoretic models. As these examples indicate, inverse optimization problems typically have two primary objectives: (a) *parameter estimation*, where we seek to infer certain parameters of a system that are consistent with a set of observations (e.g., seismic and medical tomography, revealed preference theory); and (b) *solution imposition*, where the goal is to (minimally) perturb the system parameters so as to enforce a set of solutions (e.g., the telecommunication routing application mentioned above, and network tolling where we want to find (minimal) edge tolls imposing a given routing pattern as an equilibrium flow).

Motivated by ISP, we consider inverse optimization problems wherein the desired cost vector c is required to be positive, *integral*, and induce the given subset $\mathcal{S}$ of solutions as the *unique optimal solutions* to the underlying optimization problem; we call these problems *integral inverse optimization problems.* We primarily consider the objective of minimizing $\|c\|_\infty$, but our results also yield guarantees for the objective of minimizing the perturbation $\|c - c^{(0)}\|_\infty$ of a given "base" cost vector $c^{(0)}$, which is frequently considered in the inverse-optimization literature. Uniqueness can be important because we may want to explain/impose $\mathcal{S}$ without introducing spurious solutions (i.e., "we get precisely what we bargain for"), and integrality is, in many cases, a desirable or necessary practical consideration (as in the telecommunication-routing setting). Despite extensive literature, very few algorithmic results are known for inverse optimization problems involving integrality constraints on the desired cost vector whose norm, or deviation from a given cost vector $c^{(0)}$, is to be minimized; we only know of [3, 4] that address this, both in the context of ISP.

Our contributions and results. We initiate a systematic study of integral inverse optimization problems from the perspective of designing polynomial time (approximation) algorithms. We focus on the inverse versions of various combinatorial optimization problems, a natural starting point to investigate integral inverse optimization problems. As our results demonstrate, even for such problems, wherein the underlying optimization problem is well structured and polytime-solvable, the resulting integral inverse optimization problems are

Table 1 Summary of our main results. These are stated for the implicit model, wherein the solution-set is specified implicitly by listing its support set. Most of our guarantees also hold in the explicit model.

Problem	**Our results**
Inverse shortest path (ISP)	polytime
Node-disjoint multicommodity ISP	additive 1-approximation; problem is *NP*-hard (Previous work gives multiplicative $O(\|V\|)$-approx.)
Inverse polyhedral optimization (IOpt-Poly) with TU constraint matrix	additive 1-approximation for minimization (IMin-Poly) multiplicative 2-approx. for maximization (IMax-Poly)
IOpt-Poly with $\{0,1\}$ matrix A (r, k = row, column sparsity of A)	multiplicative $\widetilde{O}\big(\sqrt{\min\{r,k\}}\big)$-approximation additive factors of k: IMin-Poly; $(k-1)$: IMax-Poly
Inverse versions of min-cost flow and min/max-cost bipartite matching	additive 1-approx.; follows from results for IOpt-Poly
Inverse matroid-basis optimization	polytime (for minimization and maximization)

quite non-trivial and exhibit an interesting range of possibilities in terms of positive (approximation) algorithmic results and hardness of approximation results. We obtain tight or nearly-tight guarantees for a variety of integral inverse optimization problems, including the well-studied inverse shortest path (ISP) problem. Our salient contributions are as follows; Table 1 summarizes our main results.

- We begin by considering ISP (Section 3). We show that the single-commodity version (Section 3.1), wherein $\mathcal{S}$ is a subset of $s \rightsquigarrow t$ paths in a directed graph, is *polytime solvable* (Theorem 5). We then consider *multicommodity* ISP, the generalization where we have multiple commodities, each specified by an (s_i, t_i) pair of nodes and a subset $\mathcal{S}_i$ of $s_i \rightsquigarrow t_i$ paths, and we seek positive, integral edge costs that ensure that $\mathcal{S}_i$ is the unique set of shortest $s_i \rightsquigarrow t_i$ paths for each commodity i. We *resolve* the status of node-disjoint multicommodity ISP, where the $\mathcal{S}_i$s correspond to node-disjoint subgraphs (Section 3.2): we devise an *additive* 1-*approximation* algorithm (Theorem 6), which is the best possible guarantee (if $P \neq NP$) since we show that this node-disjoint version is *NP*-hard (Theorem 7). Our proof also shows that it is *NP*-hard to obtain a multiplicative $\big(\frac{3}{2}-\epsilon\big)$-approximation for multicommodity ISP, for any $\epsilon > 0$.
  Our results improve upon the previous-best multiplicative $O(|V|)$-approximation guarantees for these problems, which follow from the work of [3, 4]. The algorithms in [3, 4] are for multicommodity ISP, but they apply to the restrictive setting where $\mathcal{S}_i$ includes a *single* $s_i \rightsquigarrow t_i$-path for every commodity; moreover, they do not yield improved guarantees even for the special cases of single-commodity ISP or node-disjoint multicommodity ISP. We also improve upon the factor $\frac{9}{8}$ hardness-of-approximation guarantee in [4].
- Motivated by the fact that many combinatorial optimization problems can be cast as polyhedral optimization problems, in Section 4, we consider a broad generalization of integral inverse discrete optimization, namely *integral inverse polyhedral optimization*. Here, we are given a *polytope* $\mathcal{P} \subseteq \mathbb{R}^n$ explicitly, and the set $\mathcal{S}$ is replaced by a set X of *extreme points* of $\mathcal{P}$; we seek a positive, integral cost vector $c \in \mathbb{Z}^n$ that induces X as the unique set of extreme-point optimal solutions to the problem of optimizing (minimizing or maximizing) $c^T x$ over $x \in \mathcal{P}$. We obtain approximation guarantees for integral inverse polyhedral optimization that depend on the structure of the constraint matrix A defining $\mathcal{P}$. When A is *totally unimodular* (TU), we obtain an *additive* 1-, *or multiplicative* 2-*approximation* (see Theorem 8), and for a general $\{0,1\}$ matrix A, our approximation factor depends on the *row and/or column sparsity* of A (see Theorem 9). As *corollaries*

of these results, we obtain *additive* 1-*approximation* algorithms for the integral inverse versions of min-cost flows and max/min-cost bipartite matchings.

Similar to ISP, *integral inverse min-cost flow* (IMCF) captures the optimization problem encountered in the context of *spanning-tree protocols* (STPs) – e.g., rapid STP, multiple STP etc. – which route using a *shortest-path tree* rooted at a given node s under the assigned link weights; enforcing a prescribed routing tree rooted at s by choosing small, positive, integer link lengths is then an IMCF problem, and in fact, the special case involving a single source and infinite (or equivalently, very large) capacities. This link-weight assignment problem was studied in [15, 16], who prove upper bounds on the optimum value (in a more general setting). We show that this single-source IMCF problem is polytime solvable, which implies that *we can solve this link–weight assignment problem in polynomial time.*

It is illuminating to view integral inverse polyhedral optimization (IOpt-Poly) geometrically. The set of cost vectors that yield X as extreme-point optimal solutions in $\mathcal{P}$, form a *polyhedral cone*; a cost vector in the *interior* of this cone yields X as the unique set of extreme-point optimal solutions. Thus, the goal in IOpt-Poly is to find a *shortest* (in $\|.\|_\infty$-norm) *positive, integral vector* in the interior of this cone (if one exists). Viewed from this perspective, integral inverse polyhedral optimization can be seen as a problem in the field of *geometry of numbers* and in the same vein as the important shortest-vector-problem in lattices. We believe that this geometric connection makes IOpt-Poly an appealing problem of independent interest meriting further study.

- In Section 5, we consider integral inverse matroid-basis optimization. Here, $\mathcal{S}$ is a collection of bases of a matroid, and we seek positive, integer costs on the elements under which $\mathcal{S}$ is the unique set of optimal bases. We give a polytime algorithm for this problem (Theorem 12).

Our techniques are versatile and yield results for various variants (see Section 6), including, most notably, integral inverse optimization under two other commonly considered objectives in the literature: (1) ℓ_p*-norm minimization*, where we seek to minimize $\|c\|_p$; and (2) *distance minimization*, where we seek to minimize the perturbation $\|c - c^{(0)}\|_\infty$ of an integral "base" vector $c^{(0)}$. Our results typically also hold in an *implicit* model, where the input specifies a (potentially exponential-size) set $\mathcal{S}$ implicitly by listing the elements in terms of its support.

Most prior results on inverse optimization, with the exception of ISP, are obtained in the setting where $\mathcal{S}$ consists of a single solution $\hat{x}$ (with [26, 28] being exceptions), which is not required to be the unique optimal solution, and the objective is to minimize $\|c - c^{(0)}\|_\infty$ (or $\|c - c^{(0)}\|_p$ for some other p), with c fractional. This setting is significantly simpler than the integral inverse optimization setting we consider. In particular, it is not hard to see that, as noted in [2], even for a general inverse polyhedral optimization problem, one can: (a) utilize the complementary slackness (CS) conditions from LP theory to encode the problem of finding a suitable cost vector c as another LP (or a convex program for ℓ_p norms); or (b) use the ellipsoid method to solve the LP that directly encodes that $\hat{x}$ has optimal objective value among all $x \in \mathcal{P}$, given an optimization/separation oracle for $\mathcal{P}$. This work therefore focuses on obtaining faster algorithms for the integral inverse optimization problem.

In contrast, in the integral inverse optimization setting, two distinct sources of difficulty arise that do not appear in the above setup. First, even computing a suitable fractional cost vector is non-trivial due to the uniqueness constraint. For instance, in inverse polyhedral optimization, this entails discerning if the given solutions form the extreme points of a face of the given polytope, and determining how to encode, and separate over, the constraints enforcing uniqueness. Second, rounding a fractional cost vector poses the difficulty that we need to *coordinate* things so as to simultaneously ensure that all solutions in $\mathcal{S}$ *continue to have the same cost*, and solutions not in $\mathcal{S}$ *remain non-optimal solutions.* This creates unique challenges, and we leverage tools from optimization theory, polyhedral theory, and recent results in discrepancy theory to circumvent these difficulties and obtain our results. An interesting and notable implication of our work is that, in many cases, *imposing integral costs does not significantly impact the achievable performance guarantees.*

Our array of results allude to the richness of integral inverse optimization problems. While our work makes significant progress towards understanding these problems, it also opens up various directions for further research, such as investigating the inverse-optimization versions of *NP*-hard optimization problems.

Related work. Inverse problems were initially extensively studied in geophysics for the estimation of model parameters (see, e.g., [23]). Since then there has been a great deal of work in inverse optimization in the optimization community (see, e.g., the survey [17]). In the optimization community, Burton and Toint [9] (see also [8]) were the first to consider inverse optimization problems. They introduced the the ℓ_2-norm distance-minimization variant of ISP,where we seek to minimize $\|c - c^{(0)}\|_2$, where $c^{(0)}$ is a base vector, while allowing for fractional cost vectors, and do not require the given paths to be the unique shortest paths. They motivate ISP from applications in traffic modeling and seismic tomography, and suggest the extension to the ℓ_1 and ℓ_∞ norms. Ben-Ameur and Gordin [3] and Bley [4] study (among other problems) ISP under the constraints of positive, integral edge costs, and uniqueness of the given paths (i.e., integral ISP), motivated by its applications to shortest-path routing protocols. These give algorithms having multiplicative approximation ratios of $O\big(\min\{|V|/2, (\text{maximum length of a given path})\}\big)$, and [4] also shows that it is *NP*-hard to obtain an approximation ratio better than 9/8. Other ISP variants have also been investigated [2, 4, 7, 11, 12, 13, 25].

Following initial work on inverse shortest paths, algorithms were developed for the inverse-optimization versions of other combinatorial optimization problems, such as minimum spanning tree, min-cost flow, min-cut, matroid intersection, and general inverse polyhedral optimization (also called *inverse linear programming* [29, 30]); see [17] for details. Most of this work pertains to the distance-minimization problem when we allow fractional costs, and only a single solution is given ([26, 28] are exceptions that consider multiple solutions) that is not required to be the unique optimal solution. These papers focus on developing fast combinatorial algorithms. Ahuja and Orlin [2] unify and generalize many of these results. They note that inverse polyhedral optimization can be solved in the above setting by solving a suitable LP: a compact LP encoding this can be obtained by utilizing the CS conditions, and even the (huge) LP that directly encodes that the given solution be optimal can be solved via the ellipsoid method. They show that in various cases, the compact LP leads to an LP similar to the one for optimizing over $\mathcal{P}$, and hence one can obtain combinatorial algorithms for various inverse discrete optimization problems. Similar results were also obtained by [27].

We remark that while we also solve an LP to obtain fractional cost vectors en route to obtaining integral cost vectors, a crucial difference in our setting is that we need to devise suitable ways of encoding (and separating over) the constraint that the costs induce the given (multiple) solutions as the *unique* optimal solutions. Our algorithms for integral inverse polyhedral optimization require either *a face oracle* for $\mathcal{P}$, which determines if the given set X of extreme points forms a face of $\mathcal{P}$, or an oracle that determines if all *maximal/minimal* points on a face of $\mathcal{P}$ have the same cost under a given cost vector. Devising a face oracle is related to the problem of enumerating all vertices (i.e., extreme points) of a polyhedron, or all vertices on its optimal face (under an objective function), with each new vertex being output in polynomial delay. (For instance, we can decide if X forms a face by determining if the minimal face of $\mathcal{P}$ containing X contains at least $|X|+1$ vertices.) Such procedures are known for various polyhedra such as network-flow polyhedra [20], general 0/1 *polytopes* [10], simplicial and simple polyhedra [6, 14], but this is *NP*-hard for general 0/1 polyhedra [5].

2 Problem definitions, notation, and preliminaries

For an integer n, we use $[n]$ to denote $\{1,\ldots,n\}$. Given $z \in \mathbb{R}^E$, and $S \subseteq E$, we use $z(S)$ to denote $\sum_{e\in S} z_e$. We use $\lfloor z \rfloor$ and $\lceil z \rceil$ to denote the vectors $\big(\lfloor z_e \rfloor\big)_{e\in E}$ and $\big(\lceil z_e \rceil\big)_{e\in E}$ respectively.

Inverse discrete optimization. An inverse discrete optimization problem involves an underlying discrete optimization problem specified in terms of a ground set E and a collection $\mathcal{F} \subseteq 2^E$ of feasible solutions, and a subset $\mathcal{S} \subseteq \mathcal{F}$ of feasible solutions to the optimization problem. We seek a cost vector $c \in \mathbb{R}^E$ such that the solutions in $\mathcal{S}$ are the optimal solutions to the underlying optimization problem. Formally, in an *inverse minimization problem*, the underlying optimization problem is a minimization problem, and we seek a cost vector $c \in \mathbb{R}^E$ such that $c(S) = \min_{F\in\mathcal{F}} c(F)$ for all $S \in \mathcal{S}$. In an *inverse maximization problem*, the underlying optimization problem is a maximization problem, and we seek $c \in \mathbb{R}^E$ such that $c(S) = \max_{F\in\mathcal{F}} c(F)$ for all $S \in \mathcal{S}$. More precisely, motivated by applications of the inverse-shortest-path problem in the context of shortest-path network-routing protocols in telecommunication, we impose the following requirements on the cost vector c.

(C1) *Positive, integer costs*: $c_e \geq 1,\ c_e \in \mathbb{Z}$ for all $e \in E$;

(C2) *Unique optimal solutions*: For inverse minimization, we require $c(S) = \min_{F\in\mathcal{F}} c(F) < c(F')$ for all $S \in \mathcal{S}$ and $F' \in \mathcal{F} \setminus \mathcal{S}$; for inverse maximization, we require $c(S) = \max_{F\in\mathcal{F}} c(F) > c(F')$ for all $S \in \mathcal{S}$ and $F' \in \mathcal{F} \setminus \mathcal{S}$;

Our goal is to find a vector c satisfying 1 and 2 that minimizes $\|c\|_\infty$. We call this an *integral inverse optimization problem*; we drop "integral" when it is clear from the context.

The uniqueness condition 2 is often important in applications, where the inverse optimization problem is used to infer or perturb some system parameters so as to explain or impose a set $\mathcal{S}$ of observations, since we would like to do so without introducing spurious solutions. We impose $c \geq 1$ as a normalization requirement: this prevents one from arbitrarily scaling a vector satisfying 2 to obtain another feasible solution. Integrality is a discretization condition that ensures that we are optimizing over a *closed set* (note that 2 leads to an open feasible region). (Without an underlying objective such as minimizing $\|c\|_\infty$, 1 becomes redundant as one can always scale a rational vector c to satisfy 1.)

We allow for specifying exponentially large (in the natural input size) solution sets $\mathcal{S}$ (thus obtaining greater modeling power), by also considering the following *implicit* model for specifying $\mathcal{S}$: we specify a set U of elements, which implicitly describes the set $\mathcal{S} = \{S \in \mathcal{F} : S \subseteq U\}$ of feasible solutions. For example, in the implicit version of inverse shortest paths, U is a set of arcs and $\mathcal{S}$ comprises all $s \rightsquigarrow t$ paths contained in U; so a solution is a positive, integral cost vector such that the collection of shortest $s \rightsquigarrow t$ paths is precisely $\mathcal{S}$. Our results typically apply to both models, and the underlying arguments are similar.

Our techniques are versatile and yield results for other variants of the above integral inverse optimization problem such as, most notably,

(1) the *ℓ_p-norm version*: find a vector c satisfying 1, 2 that minimizes $\|c\|_p$

(2) the *distance-minimization version* (with ℓ_∞ norm): the input specifies a "base" vector $c^{(0)} \in \mathbb{Z}_+^E$, and we seek a cost vector c satisfying 1, 2 that minimizes $\|c - c^{(0)}\|_\infty$.

At a high level, this follows because our results are obtained by first obtaining an (near-) optimal fractional cost vector c^* satisfying 1, 2 via an LP (or, for ℓ_p-norms where $1 < p < \infty$, via a convex program) and then rounding it to a feasible integral vector $\tilde{c}$ while introducing an *additive* $O(1)$ rounding error; this rounding error easily translates to a multiplicative approximation for problems (1), (2). The following theorem makes this precise.

▶ **Theorem 1.** *Let $c^* \in \mathbb{R}^E$ be a cost vector satisfying $c^*_e \geq 1\ \forall e \in E$. Let $\tilde{c} \in \mathbb{Z}^E$ be a vector satisfying 1, 2.*

(i) *Let ${O^*}_p := \min\left\{\|c\|_p : c \text{ satisfies } 1, 2\right\}$. Suppose that $\|c^*\|_p \leq {O^*}_p + \epsilon$, and $\tilde{c}_e \leq \alpha c^*_e + \beta$ for all $e \in E$. Then, $\|\tilde{c}\|_p \leq (\alpha+\beta)(1+\epsilon){O^*}_p$; if $\epsilon < \left(\frac{1}{2(\alpha+\beta){O^*}_p}\right)^p$, this implies that $\|\tilde{c}\|_p \leq \lceil \alpha+\beta \rceil\, {O^*}_p$.*

(ii) *Let $O^*_{\mathsf{dist}} := \min\left\{\|c - c^{(0)}\|_\infty : c \text{ satisfies } 1, 2\right\}$. Suppose that $O^*_{\mathsf{dist}} > 0$, $\|c^* - c^{(0)}\|_\infty \leq O^*_{\mathsf{dist}}$, and $\|\tilde{c} - c^*\|_\infty < \beta$. Then, $\|\tilde{c} - c^{(0)}\|_\infty \leq O^*_{\mathsf{dist}} + \lceil\beta\rceil - 1 \leq \lceil\beta\rceil\, O^*_{\mathsf{dist}}$.*

Inverse polyhedral optimization. Many combinatorial optimization problems have convenient polyhedral descriptions and can be modeled via linear programs that have integral optimal solutions; this indeed holds for the problems whose integral inverse optimization versions we investigate. With this in mind, we consider the following general inverse polyhedral optimization problem, which is a natural abstraction of an inverse discrete optimization problem. We are given a *polytope* $\mathcal{P} \subseteq \mathbb{R}^E_+$ with explicitly specified constraints, and a collection $X \subseteq \mathcal{P}$ of *extreme points* of $\mathcal{P}$. In *integral inverse polyhedral minimization* (IMin-Poly), we seek a cost vector $c \in \mathbb{R}^E$ that minimizes $\|c\|_\infty$ and satisfies 1, and (C2): $c^T\hat{x} = \min_{x\in\mathcal{P}} c^T x < c^T x'$ for every $\hat{x} \in X$ and every extreme point x' of $\mathcal{P}$ not in X. Similarly, in *integral inverse polyhedral maximization* (IMax-Poly), we seek $c \in \mathbb{R}^E$ that minimizes $\|c\|_\infty$ and satisfies 1, and (C2'): $c^T\hat{x} = \max_{x\in\mathcal{P}} c^T x > c^T x'$ for every $\hat{x} \in X$ and every extreme point x' of $\mathcal{P}$ not in X. If the underlying discrete optimization problem is captured by the problem of optimizing over $\mathcal{P}$ (e.g., if extreme points of $\mathcal{P}$ correspond to feasible solutions to the discrete optimization problem), then this integral inverse polyhedral optimization problem captures the integral inverse discrete optimization problem defined earlier. As before, we also consider the implicit version, wherein we are given $U \subseteq E$, which implicitly specifies $X := \big\{\text{extreme points } \hat{x} \text{ of } \mathcal{P} \text{ s.t. } \{e : \hat{x}_e > 0\} \subseteq U, \hat{x} \text{ is maximal/minimal in } \mathcal{P}\big\}$. By $\hat{x}$ being maximal in $\mathcal{P}$, we mean that there is no $x \in \mathcal{P}$ such that $x \geq \hat{x}$, $x \neq \hat{x}$; minimality is similarly defined. The set X must be maximal for IMax-Poly, and minimal for IMin-Poly, as only such points can be optimal solutions since $c > 0$.

We say that X forms a face of $\mathcal{P}$, if X is precisely the set of extreme points of some face of $\mathcal{P}$. Integral inverse polyhedral optimization can be stated geometrically as: determine if X forms a face, say F, of $\mathcal{P}$, and if so, find a positive, integral vector (if one exists) of minimum ℓ_∞ norm in the interior of the polyhedral cone of vectors yielding F as the optimal face.

Difference systems. We often need to obtain a solution to a system of constraints of the following form, called a *difference system with bounds*, involving n variables $z_1, \ldots, z_n$:

$$z_i - z_j \leq d_{ij} \quad \forall (i,j) \in A, \qquad z_i \geq \ell_i \quad \forall i \in L, \qquad z_i \leq u_i \quad \forall i \in U \tag{1}$$

where $A \subseteq [n] \times [n]$, $L, U \subseteq [n]$. The d_{ij}s can be arbitrary, so (1) can also incorporate constraints of the form $z_i - z_j \geq d_{ij}$. The following useful result is well known (see, e.g., [1]).

▶ **Theorem 2.** *We can find a feasible solution to a difference system* (1)*, or detect it is infeasible, by computing a shortest path in a digraph with $|A| + |L| + |U|$ arcs, $n+1$ nodes. If the data is integral, and* (1) *is feasible, this yields an integer-valued feasible solution.*

Further, given costs $\{b_i\}_{i=1}^n$, we can solve a min-cost flow problem to find an optimal solution to the following LP: minimize $\sum_i b_i z_i$ subject to (1)*. If this LP has an optimal solution and the d_{ij}s, ℓ_is and u_is are integral, this yields an integer-valued optimal solution.*

3 The inverse shortest path problem

In the *integral inverse shortest path* (ISP) problem, we are given a directed graph $D = (V, E)$, terminals $s, t \in V$, and a collection $\mathcal{S}$ of simple $s \rightsquigarrow t$ paths; we seek positive, integral edge costs $\{c_e\}_{e \in E}$ such that the paths in $\mathcal{S}$ are the unique shortest $s \rightsquigarrow t$ paths under these edge costs, so as to minimize $\|c\|_\infty = \max_e c_e$. In *multicommodity* ISP, we have k commodities, with each commodity $i = 1, \ldots, k$ specified by a pair $s_i, t_i \in N$ of terminals, and a collection $\mathcal{S}_i$ of $s_i \rightsquigarrow t_i$ paths. We seek positive, integral edge costs $\{c_e\}_{e \in E}$ minimizing $\|c\|_\infty$ such that for each commodity $i = 1, \ldots, k$, the paths in $\mathcal{S}_i$ are the unique $s_i \rightsquigarrow t_i$ shortest paths under these edge costs. Clearly, ISP is the special case where $k = 1$. In the implicit version of multicommodity ISP, we are given edge-sets $E^1, \ldots, E^k$, which implicitly defines $\mathcal{S}_i$ to be the collection of all $s_i \rightsquigarrow t_i$ paths in E^i.

We show that ISP is *polytime solvable* (Section 3.1). For multicommodity ISP (Section 3.2), we devise an *additive* 1-approximation algorithm in the setting where the $\mathcal{S}_i$s correspond to node-disjoint subgraphs. Our guarantee is *tight*, since we show that (even) this special case of multicommodity ISP is *NP*-hard to approximate within a factor better than $\frac{3}{2}$. Previously, only a multiplicative $O(|V|)$-approximation guarantee was known for these problems [3, 4], and a factor $\frac{9}{8}$ hardness-of-approximation was known for general multicommodity ISP [4]. In Section 6, we show that our techniques yield results for various other ISP variants including: (1) the ℓ_p-norm minimization version; (2) the distance minimization version; and (3) variants involving shortest-$s_i \rightsquigarrow t_i$-path distances in the objective or constraints.

3.1 A polynomial time exact algorithm for ISP

We may assume that every edge in D lies on some $s \rightsquigarrow t$ path, as otherwise we can assign it cost 1, and so can simply delete the edge. Let O^* denote the optimal value of the ISP instance. We utilize the following well-known properties of shortest paths.

▶ **Claim 3.** *Let $D = (N, A)$ be a digraph with nonnegative edge costs $\{c_e\}_{e \in A}$, and $s, t \in N$. Suppose that every edge of A lies on some $s \rightsquigarrow t$ path. Let $\mathcal{S}$ be a collection of $s \rightsquigarrow t$ paths.*

(i) *$\mathcal{S}$ consists of shortest $s \rightsquigarrow t$ paths (under c) iff there are* node potentials *$\{y_v\}_{v \in N}$ such that:*

$$y_v - y_u \le c_{u,v} \quad \text{for all } (u,v) \in A, \qquad y_v - y_u = c_{u,v} \quad \text{for all } (u,v) \in \bigcup_{P \in \mathcal{S}} P. \tag{2}$$

(ii) *Node potentials satisfying* (2) *exist iff the node potentials obtained by setting $y_v =$ (shortest-$s \rightsquigarrow v$-path distance) $\forall v$, satisfy* (2)*.*

(iii) *$\mathcal{S}$ comprises shortest $s \rightsquigarrow t$ paths iff every $s \rightsquigarrow t$ path $Q \subseteq \bigcup_{P \in \mathcal{S}} P$ is shortest $s \rightsquigarrow t$ path.*

If the input is in the explicit model (i.e., $\mathcal{S}$ is explicitly given), define $E^1 := \bigcup_{P \in \mathcal{S}} P$. By Claim 3 (iii), an ISP instance in the explicit model is feasible only if $\mathcal{S}$ includes all $s \rightsquigarrow t$ paths contained in E^1. Also, since we seek positive edge costs, E^1 must be acyclic (a directed cycle must have cost 0 due to (2)), otherwise the ISP instance is infeasible. In the explicit model, we first check if E^1 contains an $s \rightsquigarrow t$ path not in $\mathcal{S}$. This can be checked in polynomial time in various ways: for instance, we can use topological sort to count the number of $s \rightsquigarrow t$ paths in E^1 and check if this number is $|\mathcal{S}|$. (We can also use depth-first search and backtracking to enumerate $|\mathcal{S}|+1$ distinct $s \rightsquigarrow t$ paths in polytime (if they exist); see, e.g., [21].)

In the sequel, we assume that the ISP instance meets these feasibility requirements (so the explicit and implicit models coincide). Let $G^1 = (V^1, E^1)$ be the subgraph induced by E^1. We may assume that every edge $e \in E^1$ lies on an $s \rightsquigarrow t$ path contained in E^1 (which holds by definition in the explicit model); otherwise, we can remove e from E^1 and solve the resulting ISP instance. We consider the following LP-relaxation of the problem with the c_es and node potentials $\{y_v\}_{v\in N}$ as variables. (The objective function and constraints are easily linearized.)

$$\begin{aligned} \min \quad & \|c\|_\infty && \text{(ISP-P)}\\ \text{s.t.} \quad & \max\{1, y_v - y_u\} \le c_{u,v} \quad \forall (u,v)\in E, \qquad y_v - y_u = c_{u,v} \quad \forall (u,v)\in E^1 && (3)\\ & y_v - y_u + 1 \le c(P) \quad \forall (u,v)\in V^1\times V^1,\ \forall u \rightsquigarrow v \text{ paths } P \subseteq E\setminus E^1. && (4) \end{aligned}$$

Constraints (3) follow from Claim 3, and ensure that all $s \rightsquigarrow t$ paths in E^1 are shortest $s \rightsquigarrow t$ paths. Note that if there is no $u \rightsquigarrow v$ path in $E \setminus E^1$, then there is no constraint (4) for (u,v). We argue below that constraints (4) are valid; this follows because (4) encodes that every $s \rightsquigarrow t$ path Q not contained in E^1 has length at least $1 + \min_{s\rightsquigarrow t \text{ path } P: P\subseteq E^1} c(P)$, and with integer edge costs, this is equivalent to the condition that every $s \rightsquigarrow t$ path Q not contained in E^1 is not a shortest $s \rightsquigarrow t$ path.

▶ **Lemma 4.** (ISP-P) *is a relaxation of* ISP*.*

We can efficiently solve (ISP-P) via the ellipsoid method since we can efficiently separate over constraints (4) when $c \ge 0$ by solving a shortest-path problem. (We can actually avoid the ellipsoid method and obtain a much more efficient algorithm for ISP. We retain the LP-based exposition since this extends easily to multicommodity ISP and other variants of ISP.) If (ISP-P) is infeasible, then the ISP instance is infeasible. Otherwise, let (c^*, y^*) be an optimal solution to (ISP-P). Let $B^* = \|c^*\|_\infty$. Note that $O^* \ge \lceil B^* \rceil$. Our rounding algorithm is quite simple. We first round the $\{y^*_v\}$ node potentials by solving the following difference system:

$$\lfloor y^*_v - y^*_u \rfloor \le \psi_v - \psi_u \le \lceil y^*_v - y^*_u \rceil \qquad \text{for all } (u,v) \in V^1 \times V^1.$$

Notice that $\psi = y^*$ is a feasible solution to this difference system, so since the constant terms in the above inequalities are integers, it has a feasible integer solution $\tilde{y}$ (Theorem 2). We set edge costs $\tilde{c}_{u,v} = \tilde{y}_v - \tilde{y}_u$ for all $(u,v) \in E^1$, and $\tilde{c}_{u,v} = \lceil c^*_{u,v} \rceil$ for all $(u,v) \in E \setminus E^1$.

▶ **Theorem 5.** *Vector $\tilde{c}$ satisfies $\lfloor c^* \rfloor \le \tilde{c} \le \lceil c^* \rceil$, and is hence an optimal solution to* ISP*.*

3.2 Multicommodity ISP with node-disjoint subgraphs

We now consider multicommodity ISP, where the edges in the $\mathcal{S}_i$s induce node-disjoint subgraphs. More precisely, if the input is in the explicit model, define $E^i := \bigcup_{P\in\mathcal{S}_i} P$. Let $G^i = (V^i, E^i)$ be the subgraph induced by E^i. We consider the setting where the V^is are disjoint; we call this *node-disjoint multicommodity* ISP. As before, by Claim 3, a muticommodity ISP instance in the explicit model is feasible only if $\mathcal{S}_i$ includes all $s_i \rightsquigarrow t_i$ paths contained in E^i for all $i = 1, \dots, k$, which can be verified efficiently. Moreover, each E^i must be acyclic, and we may assume that for every i, and every $e \in E^i$, there is some $s_i \rightsquigarrow t_i$ path contained in E^i that contains e. We prove the following results, which together resolve the complexity of node-disjoint multicommodity ISP.

▶ **Theorem 6.** *There is an additive* 1*-approximation for node-disjoint multicommodity* ISP*.*

▶ **Theorem 7.** *Node-disjoint multicommodity* ISP *is* NP*-hard. Moreover, it is* NP*-hard to obtain a multiplicative* $(\frac{3}{2}-\epsilon)$*-approximation for any* $\epsilon > 0$*.*

4 Inverse polyhedral optimization

Recall that in an abstract integral inverse polyhedral optimization problem, we are given a polytope $\mathcal{P} \subseteq \mathbb{R}^E_+$ with explicitly specified constraints, and a set X of extreme points of $\mathcal{P}$. We want to find a positive, integral cost vector $c \in \mathbb{R}^E$ minimizing $\|c\|_\infty$ such that: (i) in inverse polyhedral minimization (IMin-Poly), X is the set of extreme-point optimal solutions to $\min_{x\in\mathcal{P}} c^T x$; and (ii) in inverse polyhedral maximization (IMax-Poly), X is the set of extreme-point optimal solutions to $\max_{x\in\mathcal{P}} c^T x$. In the implicit version, we are given $U \subseteq E$, which defines X to be all extreme points $\hat{x}$ of $\mathcal{P}$ such that $\{e : \hat{x}_e > 0\} \subseteq U$, and $\hat{x}$ is maximal (for IMax-Poly) or minimal (for IMin-Poly) in $\mathcal{P}$.

Our approach consists of two main steps. We first find an optimal fractional cost vector $c^* \geq 1$, and then round this. While prior work also deals with obtaining such a fractional cost vector, in our case, this step is significantly more complicated due to both the existence of *multiple* solutions in X, and the requirement that these be the *unique* optimal solutions. Let $Ax \leq b$ denote the constraints of $\mathcal{P}$ (including nonnegativity). Let K be an integer such that all entries of A, b, and all extreme points of $\mathcal{P}$ are integer multiples of $\frac{1}{K}$. We can compute K with $\log K = \text{poly(input size)}$. (If A is totally unimodular (TU) and b is integral, then $K = 1$.) So for any solution c to IMax-Poly or IMin-Poly, we have $|c^T\hat{x} - c^T x| \geq \frac{1}{K}$ for any $\hat{x} \in X$ and $x' \notin X$. For IMin-Poly, we solve the following LP-relaxation to find c^*. (For IMax-Poly, (6), and the arguments below, are modified appropriately.)

$$\text{(IMin-P)} \quad \min \quad \|c\|_\infty \quad \text{s.t.} \quad \begin{cases} c_e \geq 1 \quad \forall e \in E, \quad c^T\hat{x} = \lambda \quad \forall \hat{x} \in X & (5) \\ c^T\hat{x} \geq \lambda + \dfrac{1}{K} \quad \forall \hat{x} : \hat{x} \text{ is an extreme point of } \mathcal{P}, \hat{x} \notin X. & (6) \end{cases}$$

To solve (IMin-P) in the explicit model, we require a face oracle for $\mathcal{P}$. We first use this to determine if X forms a face F of $\mathcal{P}$; if not, then the inverse problem is infeasible. Otherwise, letting J be the set of constraints that are tight for all $x \in X$, the face F is given by $F = \{x \in \mathcal{P} : (Ax)_i = b_i \ \forall i \in J\}$. Further, any extreme point $x \in \mathcal{P} \setminus X$ does not lie in F, so there is some $i \in J$ such that $(Ax)_i < b_i$, and hence $(Ax)_i \leq b_i - \frac{1}{K}$. Our separation oracle for (IMin-P) is as follows. Constraints (5) can be directly checked. For (6), we consider every $i \in J$ and check that the minimum value of $c^T x$ over the set $\{x \in \mathcal{P} : (Ax)_i \leq b_i - \frac{1}{K}\}$ is at least $\lambda + \frac{1}{K}$. This can be done in polynomial time.

In the implicit setting, to separate over constraints (5), we require what we call a *minimality oracle* (or maximality oracle, for IMax-Poly), which given a set $U \subseteq E$ and a cost vector c, determines if every minimal (extreme) point of $\mathcal{P}$ whose support lies in U has the same (minimum) cost. To verify if constraints (6) hold, first note that if $x' \notin X$ is an extreme point supported on U, then there is some $\hat{x} \in X$ with $\hat{x} \leq x'$, and so $c^T x' \geq c^T \hat{x} + \frac{1}{K}$. So we only need to check if (6) holds for extreme points x' not supported on U; this can be done by the same procedure as for the explicit model, taking $J = E \setminus U$. The problem of devising a minimality/maximality oracle is itself an interesting and non-trivial problem for various combinatorial optimization problems. We show how to devise such an oracle for min-cost flow and bipartite matching (Theorems 10 and 11). Note that for a *polytope* $\mathcal{P}$ of the form $\{x : Ax = b, \ x \geq 0\}$, any two feasible points are incomparable; so the face formed by X is simply $F = \{x \in \mathcal{P} : x_e = 0 \ \forall e \notin U\}$, and we can obtain a minimality/maximality oracle by checking if the minimum and maximum values of $c^T x$ over F are equal.

The next step is to round c^* to obtain an approximately optimal integral cost vector. We show how to do this in two settings, when the constraint matrix A is TU, and when A is a sparse $\{0,1\}$-matrix. In both cases, we round an optimal solution to the dual of the problem of optimizing ${c^*}^T x$ over $\mathcal{P}$ but the details and bounds obtained differ.

▶ **Theorem 8.** *Let $\mathcal{P} = \{x \in \mathbb{R}^E : Ax \geq b,\ x \geq 0\}$, where A is TU, and b is integral. We can round an optimal fractional solution c^* to the inverse problem to obtain: (a) an additive 1-approximation for* IMin-Poly*; and (b) a multiplicative 2-approximation for* IMax-Poly*.*

▶ **Theorem 9.** *Let $A \in \{0,1\}^{m \times n}$ have row sparsity r and column sparsity k, where $n = |E|$. (Row sparsity is the maximum number of nonzero entries in a row of A; column sparsity is the maximum number of nonzero entries in a column of A.) Let A_1 and A_2 be submatrices of A with n columns, whose rows partition $[m]$. We can round an optimal fractional solution c^* to the inverse problem to obtain the following guarantees (in both implicit and explicit models).*

(a) *Additive $(k-1)$-approx. for* IMax-Poly *with $\mathcal{P} = \big\{x \in \mathbb{R}^E : A_1 x = b_1,\ A_2 x \leq b_2,\ x \geq 0\big\}$.*

(b) *Additive k-approximation for* IMin-Poly *with $\mathcal{P} = \big\{x \in \mathbb{R}^E : A_1 x = b_1,\ A_2 x \geq b_2,\ x \geq 0\big\}$.*

(c) *Multiplicative β-approximation for* IMax-Poly *and* IMin-Poly*, where we have $\beta = \min\big\{k + O(1), O(\sqrt{r \log n}), O(\sqrt{k}\min(\log(kr), \log n))\big\}$.*

4.1 Applications to inverse min-cost flow and inverse bipartite matching

Inverse min-cost flow. In the *integral inverse min-cost flow* (IMCF) problem, we are given a directed graph $D = (N, E)$, integer bounds $0 \leq \ell_e \leq u_e$ on every edge e, integer demands $\{b_v\}_{v \in N}$ (which could be arbitrary) such that $b(N) := \sum_{v \in N} b_v = 0$, and a set $E^1 \subseteq E$ of edges. A *flow* in D is a vector $x \in \mathbb{R}^E$ satisfying

$$x\big(\delta^{\text{in}}(v)\big) - x\big(\delta^{\text{out}}(v)\big) = b_v \quad \forall v \in N, \qquad \ell_e \leq x_e \leq u_e \quad \forall e \in E. \tag{7}$$

Given edge costs $\{c_e\}_{e \in E}$, the cost of a flow x is $\sum_e c_e x_e$. We seek positive, integral edge costs $\{c_e\}_{e \in E}$ minimizing $\|c\|_\infty$ so that the set of min-cost integral flows is precisely the set of acyclic integral flows supported on E^1. As with ISP, we may assume that every $e \in E^1$ is used by some feasible flow supported on E^1, and then, we may further assume that E^1 is acyclic, as otherwise the inverse problem is infeasible.

The min-cost flow problem is given by the LP: min $\sum_e c_e x_e$ subject to (7). The constraint matrix specifying (7) is TU (see, e.g., [22]), so IMCF is an instance of IMin-Poly with a TU constraint matrix. Since E^1 is acyclic, any two distinct feasible flows supported on E^1 are incomparable, so it is easy to obtain a minimality oracle and solve (IMin-P). We then obtain the following positive result in the above implicit model as a corollary of Theorem 8 (a). We discuss the explicit model in the full version. where we also show that IMCF is polytime solvable in certain cases, such as, the single-source setting with no (or equivalently, very large) capacities. As noted earlier, in the context of *spanning-tree protocols*, this implies that *in polynomial time, we can find the smallest positive integer link weights that enforce a prescribed routing tree as a shortest-path tree rooted a given node s.*

▶ **Theorem 10.** *There is an additive 1-approximation for* IMCF *in the implicit model.*

Inverse bipartite matching. In inverse bipartite matching, the input is an undirected bipartite graph $G = (V, E)$. In *integral max-cost bipartite matching* (IMax-BMat), we have a collection $M_1, \ldots, M_k$ of maximal matchings, and we seek positive, integral edge costs $\{c_e\}_{e \in E}$

minimizing $\|c\|_\infty$ so that $M_1, \ldots, M_k$ are the unique max-cost bipartite matchings in G. In the implicit model, we are given $E^1 \subseteq E$, and we require that the set of max-cost bipartite matchings be the set of maximal matchings contained in E^1. (Max-cost matchings must be maximal.) The max-cost bipartite matching LP is: max $\sum_e c_e x_e$ s.t. $x\big(\delta(v)\big) \leq 1\ \forall v \in V,\ \ x \geq 0$.

We also consider *integral min-cost bipartite matching* (IMin-BMat), where we are given *perfect matchings* $M_1, \ldots, M_k$, and we seek positive integral edge costs $\{c_e\}_{e \in E}$ minimizing $\|c\|_\infty$ such that these are the unique min-cost perfect matchings in G. In the implicit model, we are given $E^1 \subseteq E$, and the set of min-cost perfect matchings should be the set of perfect matchings contained in E^1. The min-cost perfect matching problem can be modeled by the following LP: min $\sum_e c_e x_e$ s.t. $x\big(\delta(v)\big) = 1\ \forall v \in V,\ \ x \geq 0$. The constraint matrix in the above LPs is TU and has column sparsity 2. We devise a face oracle for IMax-BMat and IMin-BMat in the explicit setting, and a maximality oracle for IMax-BMat in the implicit setting when $E^1 = E$ by exploiting various structural properties of bipartite matchings. (A minimality oracle for IMin-BMat is easy since the corresponding polytope is defined by equations and nonnegativity constraints.) The maximality oracle for IMax-BMat determines if there exist two maximal matchings of different costs; we note that the related problem of finding a min-cost maximal matching is *NP*-hard. Theorems 8(a) and 9(a) then yield the following.

▶ **Theorem 11.** *We can obtain additive guarantees of* 1 *for* IMin-BMat, *and* IMax-BMat *in the explicit setting, and* IMax-BMat *in the implicit setting when* $E^1 = E$.

5 Inverse matroid-basis optimization

We consider the *integral inverse min-cost matroid basis* (IMin-Basis) and *integral inverse max-cost matroid basis* (IMax-Basis) problems. In both problems, the input is a matroid $M = (E, \mathcal{I})$ (specified by an independence oracle) and a collection $\mathcal{S}$ of bases of M. The goal is to find positive, integral costs $\{c_e\}_{e \in E}$ such that the bases in $\mathcal{S}$ are the unique optimal bases under these costs, so as to minimize $\|c\|_\infty$. More precisely, in IMin-Basis, we require that the bases in $\mathcal{S}$ be the unique *min-cost* bases under the $\{c_e\}$ costs, while in IMax-Basis, we require that the bases in $\mathcal{S}$ be the unique *max-cost* bases under the $\{c_e\}$ costs. In the implicit model, we are given $U \subseteq E$, which implicitly specifies $\mathcal{S}$ to be all bases of M contained in U.

▶ **Theorem 12.** *We can solve* IMin-Basis *and* IMax-Basis *in polynomial time.*

6 Extensions and variants

Our techniques are versatile and yield guarantees for other variants of integral inverse optimization mentioned in Section 2, including the ℓ_p-norm version (minimize $\|c\|_p$) and distance-minimization version (minimize $\|c - c^{(0)}\|_\infty$, where $c^{(0)} \in \mathbb{Z}_+^E$) problems; for these two problems our guarantees follow by simply combining our earlier results with Theorem 1.

Inverse shortest paths. We obtain multiplicative guarantees of 2 and 3 respectively for the ℓ_p-norm variant of ISP and multicommodity ISP respectively, and obtain the optimal solution and an additive guarantee of 1 for the distance-minimization variants. Bley [4] considered the ISP variant where we seek positive, integral costs so as to minimize $\max_{i=1,\ldots,k}$ (shortest-$s_i \rightsquigarrow t_i$-path distance). A related variant specifies integer upper bounds $\{D_i\}_{i=1}^k$ on the shortest-$s_i \rightsquigarrow t_i$-path distances, and seeks a positive, integral cost vector c that respects these bounds and minimizes $\|c\|_\infty$. The guarantees in Theorems 5, 6 hold for both variants.

▶ **Theorem 13.** *We obtain the following multiplicative guarantees for the ℓ_p-norm and distance-minimization versions of integral inverse polyhedral optimization.*

(a) 3*-approximation for* IMin-Poly *with* $\mathcal{P} = \{x \in \mathbb{R}^E : Ax \geq b,\ x \geq 0\}$*, where A is TU and b is integral.*

(b) $(k+1)$*-approximation for* IMax-Poly *with* $\mathcal{P} = \left\{x \in \mathbb{R}^E : A_1x = b_1,\ A_2x \leq b_2,\ x \geq 0\right\}$*, where* $A^T = \begin{pmatrix} A_1^T & A_2^T \end{pmatrix}$ *is a* $\{0,1\}$ *matrix, and A has column sparsity k.*

(c) $(k+2)$*-approximation for of* IMin-Poly *with* $\mathcal{P} = \left\{x \in \mathbb{R}^E : A_1x = b_1,\ A_2x \geq b_2,\ x \geq 0\right\}$*, where* $A^T = \begin{pmatrix} A_1^T & A_2^T \end{pmatrix}$ *is a* $\{0,1\}$ *matrix, and A has column sparsity k.*

In the explicit model, we assume that we have a face oracle for $\mathcal{P}$. In the implicit model, we assume that we have a minimality/maximality oracle for $\mathcal{P}$.

▶ **Theorem 14.** *(a) There is a multiplicative* 2*-approximation algorithm for the ℓ_p-norm minimization versions of* IMin-Basis *and* IMax-Basis*. (b) The distance-minimization versions of* IMin-Basis *and* IMax-Basis *can be solved exactly in polytime.*

References

1 Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network flows: theory, algorithms, and applications*. Prentice hall, 1993.

2 R.K. Ahuja and J.B. Orlin. Inverse optimization. *Oper. Res.*, 49:171–783, 2001.

3 W. Ben-Ameur and E. Gourdin. Internet routing and related topology issues. *SIAM J. Discrete Math.*, 17:18–49, 2004.

4 A. Bley. Inapproximability results for the inverse shortest paths problem with integer lengths and unique shortest paths. *Networks*, 50:29–36, 2007.

5 Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Hans Raj Tiwary. The negative cycles polyhedron and hardness of checking some polyhedral properties. *Annals of Operations Research*, 188(1):63–76, 2011.

6 David Bremner, Komei Fukuda, and Ambros Marzetta. Primal-dual methods for vertex and facet enumeration. *Discrete & Computational Geometry*, 20(3):333–357, 1998.

7 D. Burton, W. Pulleyblank, and Ph. L. Toint. The inverse shortest paths problem with upper bounds on shortest paths costs. In *Network optimization*, pages 156–171. Springer, 1996.

8 D Burton and Ph L Toint. On the use of an inverse shortest paths algorithm for recovering linearly correlated costs. *Mathematical Programming*, 63(1):1–22, 1994.

9 D. Burton and Ph.L. Toint. On an instance of the inverse shortest paths problem. *Math. Program.*, 53:45–61, 1992.

10 Michael R Bussieck and Marco E Lübbecke. The vertex set of a 01-polytope is strongly p-enumerable. *Computational Geometry*, 11(2):103–109, 1998.

11 Mikael Call and Kaj Holmberg. Complexity of inverse shortest path routing. In *INOC*, pages 339–353. Springer, 2011.

12 Robert B Dial. Minimal-revenue congestion pricing part i: A fast algorithm for the single-origin case. *Transportation Research Part B: Methodological*, 33(3):189–202, 1999.

13 Robert B Dial. Minimal-revenue congestion pricing part ii: An efficient algorithm for the general case. *Transportation Research Part B: Methodological*, 34(8):645–665, 2000.

14 Martin E Dyer. The complexity of vertex enumeration methods. *Mathematics of Operations Research*, 8(3):381–402, 1983.

15 F. Grandoni, G. Nicosia, G. Oriolo, and L. Sanità. Stable routing under the spanning tree protocol. *Operations Research Letters*, 38:399–404, 2010.

16 N. Haehnle, L. Sanità, and R. Zenklusen. Stable routing and unique-max coloring on trees. *SIAM J. Discret. Math*, 27:109–125, 2013.

17 C. Heuberger. Inverse combinatorial optimization: A survey on problems, methods, and results. *J. Combin. Optim.*, 8:329–361, 2004.

18 Garud Iyengar and Wanmo Kang. Inverse conic programming with applications. *Operations Research Letters*, 33(3):319–330, 2005.

19 Gertrud Neumann-Denzau and Jörn Behrens. Inversion of seismic data using tomographical reconstruction techniques for investigations of laterally inhomogeneous media. *Geophysical Journal International*, 79(1):305–315, 1984.

20 J. Scott Provan. Efficient enumeration of the vertices of polyhedra associated with network LP's. *Math. Program.*, 63(1):47–64, 1994.

21 Robert C Read. Bounds on backtrack algorithms for listing cycles, paths and spanning trees. *Networks*, 5:237–252, 1975.

22 Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998.

23 Albert Tarantola. *Inverse problem theory and methods for model parameter estimation.* SIAM, 2005.

24 H.R. Varian. Revealed preference. *Samuelsonian Economics and the Twenty-first Century*, pages 99–115, 2006.

25 S. Xu and J. Zhang. An inverse problem of the weighted shortest path problem. *Japan J. Indust. Appl. Math.*, 12:47–59, 1995.

26 J. Zhang and M.C. Cai. Inverse problem of minimum cuts. *Mathematical Methods of Oper. Res.*, 47:51–58, 1998.

27 J. Zhang and Z. Liu. A general model of some inverse combinatorial optimization problems and its solution method under l-infinity norm. *J. Combin. Optim.*, 6:207–227, 2002.

28 J. Zhang and Z. Ma. Solution structure of some inverse combinatorial optimization problems. *J. Combin. Optim.*, 3:127–139, 1999.

29 Jianzhong Zhang and Zhenhong Liu. Calculating some inverse linear programming problems. *Journal of Computational and Applied Mathematics*, 72(2):261–273, 1996.

30 Jianzhong Zhang and Zhenhong Liu. A further study of inverse linear programming problems. *Journal of Computational and Applied Mathematics*, 106:345–359, 1999.

Two-Dimensional Maximal Repetitions

Amihood Amir[1]
Bar Ilan University, Ramat-Gan, 52900, Israel
amir@esc.biu.ac.il

Gad M. Landau[2]
University of Haifa, Haifa 31905, Israel, and
NYU Tandon School of Engineering, New York University,
Six MetroTech Center, Brooklyn, NY 11201, USA
landau@univ.haifa.ac.il

Shoshana Marcus
Kingsborough Community College of the City University of New York
2001 Oriental Boulevard, Brooklyn, NY 11235, USA
shoshana.marcus@kbcc.cuny.edu

Dina Sokol[3]
Brooklyn College of the City University of New York
2900 Bedford Avenue, Brooklyn, NY, 11210, USA
sokol@sci.brooklyn.cuny.edu

Abstract

Maximal repetitions or *runs* in strings have a wide array of applications and thus have been extensively studied. In this paper, we extend this notion to 2-dimensions, precisely defining a *maximal 2D repetition*. We provide initial bounds on the number of maximal 2D repetitions that can occur in a matrix. The main contribution of this paper is the presentation of the first algorithm for locating all maximal 2D repetitions in a matrix. The algorithm is efficient and straightforward, with runtime $O(n^2 \log n \log\log n + \rho \log n)$, where n^2 is the size of the input, and ρ is the number of 2D repetitions in the output.

2012 ACM Subject Classification Mathematics of computing → Combinatorics on words, Theory of computation → Design and analysis of algorithms

Keywords and phrases pattern matching algorithms, repetitions, periodicity, two-dimensional

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.2

1 Introduction

Repetitions in strings constitute one of the most fundamental areas of string combinatorics. They are exploited in the design of efficient algorithms for string matching, data compression, and analysis of biological sequences. Maximal repetitions are important structures, as they encode all of the repetitions in the string in a concise way. Once the set of maximal repetitions is known, repetitions of any other type (such as squares and cubes) can be extracted from it.

[1] Partially supported by the Israel Science Foundation grant 571/14 and Grant No. 2014028 from the United States-Israel Binational Science Foundation (BSF).
[2] Partially supported by the Israel Science Foundation grant 571/14 and Grant No. 2014028 from the United States-Israel Binational Science Foundation (BSF).
[3] Partially supported by Grant No. 2014028 from the United States-Israel Binational Science Foundation (BSF).

© Amihood Amir, Gad M. Landau, Shoshana Marcus, and Dina Sokol;
licensed under Creative Commons License CC-BY

Driven by the many applications to pattern recognition, low level image processing, computer vision and multimedia, the past decades have seen the extension of clever string searching techniques and combinatorial properties to two-dimensional arrays. However, the notion of *maximal* two-dimensional repetitions has not been explored, neither from the combinatorial perspective nor from the algorithmic perspective. Thus, in this project we propose to fill this void. We define a maximal 2D repetition to be a submatrix that can be decomposed into repeating non-overlapping occurrences of the same subblock horizontally and vertically that is maximally extended in all directions.

A range of motivating applications exist that can spur the exploration of maximal repetitions in matrices. In one-dimension, algorithms that compute all the maximal repetitions in a text have application to data compression. The discovery of repetitive structures in the two-dimensional sense can lead to improvements in the compression schemes used for images and video. Just as properties of repetitions have enabled the speeding up of one-dimensional pattern searching algorithms and are relied on by space-efficient one-dimensional pattern matching algorithms, discovering properties of two-dimensional repetitions should create new possibilities and opportunities to speed up two-dimensional string matching algorithms and to design algorithms that use less working space in memory.

As Crochemore et al. have pointed out [9], "the difficulties in extending string-matching techniques to image pattern matching methods are essentially due to different and more complex structures of 2D-periodicities."

In this paper we define two-dimensional *maximal repetitions* for matrices, prove upper bounds on the number of maximal repetitions that can occur in a matrix, and develop an efficient algorithm for locating them. We begin by putting our work in context of related work in Section 2. In Section 3 we precisely define a 2D maximal repetition. Then, in Section 4, we prove that there are at most $O(n^3)$ maximal 2D repetitions in an $n \times n$ matrix. In Section 5 we develop an algorithm to find all the maximal 2D repetitions in an $n \times n$ matrix in close to linear time.

2 Related Work

A string r is *periodic* if its longest prefix that is also a suffix is at least half the length of r. A string s is *primitive* if it cannot be expressed in the form $s = u^j$, for some integer $j > 1$ and some prefix u of s. A periodic string r can be expressed as $u^j u'$ for one unique primitive u, which is called *the period* of r. Every non-primitive string is periodic but not every periodic string is non-primitive. For example, `abc, abcab` are both primitive and non-periodic, `abcabc` is non-primitive (and hence periodic), while `abcabca` is primitive and periodic with period `abc`.

In a string s, a *maximal repetition*, or *run*, is a periodic substring r with period u in which an extension by one letter to the right or to the left yields a string with a longer period than $|u|$ [16]. The maximal repetitions in a string can overlap, be embedded one within another, or begin at the same position. Thus, it was remarkable when Kolpakov and Kucherov proved that a string of length n can contain only $O(n)$ runs [16]. More recently, Bannai et al. proved that the number of runs is strictly less than n [6].

A *square* is a particular type of repetition. In one-dimension, a square is a string which consists of precisely two consecutive occurrences of a substring. Apostolico and Brimkov [3] extend the notion of a square to two dimensions, to form a 2D tandem. They define a 2D tandem as a configuration consisting of two occurrences of the same primitive block that share a side or a corner. A primitive array is one that cannot be partitioned into non-overlapping replicas of some block W [3]. Apostolico and Brimkov prove combinatorially that an $n \times n$ matrix can contain $\Theta(n^4)$ corner-sharing tandems and $\Theta(n^3 \log n)$ side-sharing tandems [3]. They develop an $O(n^3 \log n)$ algorithm for finding side-sharing tandems in an $n \times n$ matrix, which can be used to derive an $O(n^4)$ algorithm for locating all corner-sharing tandems [4].[4] In this paper we extend Apostolico and Brimkov's concept of a side-sharing 2D tandem to many copies to form maximal tandems horizontally and vertically.

A combinatoric construct that is related to repetitions is that of periodicities, i.e. highly repetetive subblocks. The different kinds of two-dimensional periodicities in matrices have been studied by Amir and Benson [1] in terms of self-overlap. Their definition of line and radiant periodicity do not result in 2D repetitions since only the overlapping portion repeats. The lattice periodicity of Amir and Benson is most similar to a 2D repetition. It is also similar in concept to the bi-periodic infinite pictures studied by Bacquey [5]. Bacquey provides interesting combinatoric properties of the primitive roots of bi-periodic infinite pictures. The current paper is more restrictive in terms of lattice periodicity in that the primitive root always has to occur immediately adjacent to its neighbor to the right or beneath it, forming a lattice with all right angles. Apostolico and Brimkov [3], at the beginning of the above-mentioned paper on tandems, define exactly this kind of repetition.

The right-angle lattice periodicity is also used by Gamard and Richomme [11] where the primitive roots of 2D arrays are studied. A matrix is defined as *primitive* if it cannot be broken down to a repeating factor vertically and/or horizontally. Gamard et al. [12] show that every matrix has one unique primitive root. They present several 2D generalizations of the Lyndon-Schutzenberger periodicity theorem for words. However, all exponents in their periodic matrices are integers, i.e. only whole copies of the primitive root are allowed in a repetitive matrix.

In this paper we discuss periodicity where partial copies are allowed at the ends of the matrix, i.e. we use *real* exponents. Our goal is to find *maximal* rectangular submatrices that are repetitions in a given matrix. In the next section we precisely define a 2D repetition and a *maximal 2D repetition* in a matrix.

3 Definition of 2D Maximal Repetition

3.1 1D Maximal Repetitions

In one-dimensional data, a maximal repetition is a substring that is a repetition such that its extension by one character to the right or to the left yields a word with a larger period [16].

▶ **Definition 1.** Let T be a 1D repetition of length t with period U of length u. The exponent e of T is the rational number that satisfies $e = \frac{t}{u}$.

▶ **Lemma 2.** *Let T be a 1D repetition of length t with period U of size u. Let the exponent e be the number of adjacent times U occurs in T such that $U^e = T$ and $u \cdot e = t$. Then T is* maximal *iff it is a substring in which extending one character to the right or left yields a string T' of size $t + 1$ with period U' of size u' and exponent e' such that $e' < e$.*

Proof. The proof has been omitted due to lack of space. ◀

[4] They consider this optimal based on the largest number of such repetitions that can occur in a matrix. However, this is not optimal for a matrix with few 2D tandems. A truly optimal algorithm would find all 2D tandems in $O(n^2 + occ)$ time, where occ is the number of 2D tandems in the matrix.

Table 1 Non-primitive matrices.

X
X

X	X

X	X
X	X

3.2 2D Maximal Repetitions

We say that U is a horizontal prefix (resp. suffix) in matrix M if U is an initial (resp. ending) sequence of contiguous columns in M. A *horizontal border* of matrix M is a proper horizontal prefix that is also a horizontal suffix of M. We say that B is the *longest horizontal border* of M if it is the horizontal border of M that spans the largest number of columns among the horizontal borders of M.

▶ **Definition 3.** The *horizontal period*, or *h-period*, of an $m \times n$ matrix M is $n - b$ where b is the number of columns contained in the longest horizontal border of M.

▶ **Definition 4.** [8, 17] An $m \times n$ matrix M with *h-period* p is *horizontally periodic*, or *h-periodic*, if $p \leq \lfloor \frac{n}{2} \rfloor$.

The vertical period of a matrix and vertical periodicity are defined analogously.

▶ **Definition 5.** [3] A matrix M is a *two-dimensional repetition* if M is h-periodic and v-periodic.

Consider an $m \times n$ matrix M and rational numbers $x > 0$, $y > 0$. $M^{x,y}$ is the matrix constructed by repeating M x times vertically and y times horizontally, yielding an $\lfloor xm \rfloor \times \lfloor yn \rfloor$ matrix.

For example,

$$M = \begin{bmatrix} a & b & c & d \\ e & f & g & h \end{bmatrix}$$

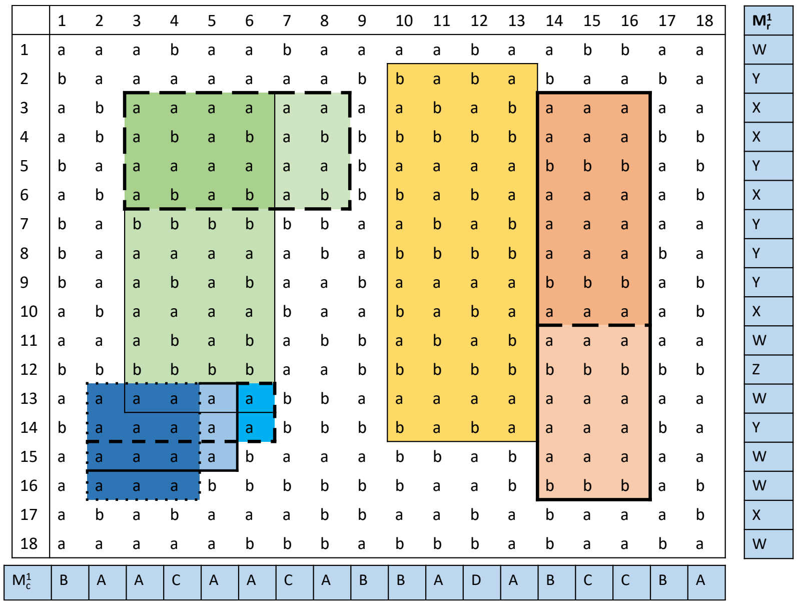
$$M^{1.5,2.25} = \begin{bmatrix} a & b & c & d & a & b & c & d & a \\ e & f & g & h & e & f & g & h & e \\ a & b & c & d & a & b & c & d & a \end{bmatrix} \quad M^{2,1.5} = \begin{bmatrix} a & b & c & d & a & b \\ e & f & g & h & e & f \\ a & b & c & d & a & b \\ e & f & g & h & e & f \end{bmatrix}$$

▶ **Definition 6.** [3, 12] A matrix M is *primitive* if it cannot be partitioned into more than one non-overlapping complete occurrences of some block W. M is *non-primitive* if M can be expressed as $M = W^{r,s}$ for integers r, s such that either $r > 1$ or $s > 1$ or both r and s are strictly greater than 1.

Table 1 shows the different basic configurations of a non-primitive matrix. As in the string terminology, a periodic matrix can be either primitive or non-primitive. In the example, $M^{1.5,2.25}$ is both periodic and primitive, while $M^{2,1.5}$ is periodic and non-primitive.

▶ **Definition 7.** The *primitive root* W of a matrix M is a primitive submatrix such that $M = W^{r,s}$ for rational numbers r, s. M begins with W at its upper left corner and can be partitioned into non-overlapping replicas of W, possibly including partial occurrences of W at its right and / or lower ends.

▶ **Lemma 8.** *Every matrix M has a unique primitive root W such that $M = W^{r,s}$ for rational numbers r, s.*



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	M_r^1
1	a	a	a	b	a	a	b	a	a	a	a	b	a	a	b	b	a	a	W
2	b	a	a	a	a	a	a	a	b	b	a	b	a	b	a	a	b	a	Y
3	a	b	a	a	a	a	a	a	a	a	b	a	b	a	a	a	a	a	X
4	a	b	a	b	a	b	a	b	b	b	b	b	b	a	a	a	b	b	X
5	b	a	a	a	a	a	a	a	b	a	a	a	a	b	b	b	a	b	Y
6	a	b	a	b	a	b	a	b	b	b	a	b	a	a	a	a	a	b	X
7	b	a	b	b	b	b	b	b	a	a	b	a	b	a	a	a	a	a	Y
8	b	a	a	a	a	a	a	b	a	b	b	b	b	a	a	a	b	a	Y
9	b	a	a	b	a	b	a	a	b	a	a	a	a	b	b	b	a	b	Y
10	a	b	a	a	a	a	b	a	a	b	a	b	a	a	a	a	a	b	X
11	a	a	a	b	a	b	a	b	b	a	b	a	b	a	a	a	a	a	W
12	b	b	b	b	b	b	a	a	b	b	b	b	b	b	b	b	b	b	Z
13	a	a	a	a	a	a	b	b	a	a	a	a	a	a	a	a	a	a	W
14	b	a	a	a	a	a	b	b	b	b	a	b	a	a	a	a	b	a	Y
15	a	a	a	a	a	b	a	a	a	b	b	a	b	a	a	a	a	a	W
16	a	a	a	a	b	b	b	b	b	b	a	a	b	b	b	b	a	b	W
17	a	b	a	b	a	a	a	b	b	a	a	b	a	b	a	a	a	b	X
18	a	a	a	a	a	b	b	b	a	b	b	b	a	b	a	a	b	a	W
M_c^1	B	A	A	C	A	A	C	A	B	B	A	D	A	B	C	C	B	A	

Figure 1 A matrix M with many maximal 2D repetitions highlighted. The first row of M_c^1 is depicted below M and the first column of M_r^1 is depicted on the right.

Proof. The proof has been omitted due to lack of space. ◀

▶ **Definition 9.** Let R be a 2D repetition of size $r_1 \times r_2$ with primitive root W of size $w_1 \times w_2$. The *exponent* of R is a tuple (e_1, e_2) in which e_1 and e_2 are rational numbers that satisfy $e_1 = \frac{r_1}{w_1}$ and $e_2 = \frac{r_2}{w_2}$.

In a 2D repetition $R =$

$$\begin{array}{|ccc|c|} \hline W & \dots & W & W' \\ \dots & \dots & \dots & \dots \\ W & \dots & W & W' \\ \hline W'' & \dots & W'' & W''' \\ \hline \end{array}$$

there are at least two W-blocks horizontally and vertically. That is, the primitive root W repeats both to the right and underneath its initial occurrence in R.

We introduce the idea of a maximal two-dimensional repetition. A 2D repetition R with root W is *maximal* if it cannot be extended by one row or one column to obtain a 2D repetition with the same primitive root W. Figure 1 depicts a matrix with many 2D maximal repetitions highlighted.

▶ **Lemma 10.** *A 2D repetition R of size $r_1 \times r_2$ with root W of size $w_1 \times w_2$ and exponent (e_1, e_2) is maximal iff extending R by one row or column in either direction yields a matrix R' of size $r'_1 \times r'_2$ with primitive root W' of size $w'_1 \times w'_2$ and exponent (e'_1, e'_2) such that $e'_2 < e_2$ or $e'_1 < e_1$.*

Proof. The proof has been omitted due to lack of space. ◀

4 Bounds on the Number of 2D Maximal Repetitions

▶ **Lemma 11.** *There are $O(n^3)$ maximal 2D repetitions in an $n \times n$ matrix.*

Proof. In each row there are $O(n)$ maximal 1D repetitions [16]. For each possible height $0 < h \leq n$, we can linearize the 2D submatrix beginning in each row with height h and width n, by naming metacharacters of subcolumns of height h. This linearization yields a string of length n with $O(n)$ runs. Thus, beginning in each row, for each height, we have $O(n)$ 2D h-periodic horizontally maximal repetitions, resulting in $O(n^3)$ over all rows. The number of 2D maximal repetitions is no more than the number of h-periodic submatrices, since all 2D maximal repetitions are h-periodic. ◀

5 Algorithm to Find 2D Maximal Repetitions

In this section we develop an efficient algorithm to identify all maximal 2D repetitions in an $n \times n$ matrix M. The naive algorithm can examine each of the $O(n^4)$ submatrices in M. For each submatrix S, we can check whether S can possibly be the primitive root of a 2D repetition by attempting to extend it as far as possible. This would take $O(n^6)$ time for all submatrices S. Using LCA queries within each row or column to extend S would speed up the algorithm to $O(n^5)$ time. The last step that remains is to filter out repetitions that were located more than once, which can complete the process in $O(n^5)$ time. The remainder of this section presents a more efficient $O(n^2 \log n \log\log n + \rho \log n)$ algorithm for finding all ρ maximal 2D repetitions that occur in M.

Algorithm Overview:

Step 1 Preprocess the matrix and set up data structures that are used later on by algorithm.

Step 2 Search in each row of the matrix for h-periodic submatrices of height 2^i, for every $1 \leq i \leq \log n$, that begin in that row.

Step 3 Locate all maximal 2D repetitions of height $2^i \leq r < 2^{i+1}$, for every $1 \leq i \leq \log n$, whose prefix 2^i rows are v-periodic.

Step 4 Identify the maximal 2D repetitions of height $2^i < r < 2^{i+1}$, for every $1 \leq i \leq \log n$, whose v-period is not apparent in the first 2^i rows.

5.1 Step 1: Preprocessing the matrix

There are three steps to the preprocessing stage of our algorithm:

1. Naming
   We use Karp-Miller-Rosenberg (KMR) naming [14] on matrix M. One-dimensional KMR naming works with a string, naming each substring whose length is a power of 2. In two-dimensions, we name subcolumns of an $n \times n$ matrix M spanning a number of rows that are powers of 2, i.e., $r = 2^i, 1 \leq i \leq \log n$. We construct $\log n$ matrices of names called M_c^i, for each $1 \leq i \leq \log n$, by naming subcolumns of heigth 2^i. Similarly, we construct a second set of $\log n$ matrices of names which we call M_r^j, for each $1 \leq j \leq \log n$, by naming subrows of M whose widths are 2^j. Throughout the rest of this paper, i is used as the exponent when denoting a number of *rows*, e.g. height 2^i, while j is used in reference to columns, e.g. width 2^j.
2. Substring Periodicity Queries
   A Substring Periodicity Query (SPQ) is as follows: given a string T of length n and two indices, $1 \leq i < j \leq n$, return the period length of $T[i..j]$, when $T[i..j]$ is a repetition. Kociumaka et al. [15] presented an algorithm that processes a string in linear time and space to support $O(1)$ time Substring Periodicity Queries, which they call 2-Period Queries. (Similar time and space complexities are presented by Bannai et al. [7].) We preprocess each column of M_r^j, for each $1 \leq j \leq \log n$, in linear time following the algorithm of Kociumaka et al. [15] to support $O(1)$ time SPQ.

3. Vertical Squares Preprocessing
   We build and decorate suffix trees of each column in each of M_r^j, $1 \leq j \leq \log n$, using the approach of Gusfield and Stoye [13]. The decorated suffix tree marks the endpoints of tandem repeats, either at a node or along an edge. In each decorated suffix tree, we add a link at each node that points to its closest ancestor that is marked or has a marked edge leading into it. We also add links from each marked node to its closest marked ancestor. We then preprocess each decorated suffix tree to admit $O(\log \log n)$ time weighted ancestor queries, where the weight of a node corresponds to the string length it encodes in the suffix tree [2].

Time Complexity for Preprocessing: Subcolumns and subrows can be named with generalized suffix trees of the matrix columns and of the matrix rows for each of the $\log n$ matrices of names. This takes $O(n^2 \log n)$ time, since the naming can be done during Ukkonen's suffix tree construction process [18]. We build a suffix tree of the matrix and preprocess it in linear time to admit $O(1)$ time LCA queries later on. The preprocessing of Kociumaka et al. [15] for SPQ runs in linear time per column of each matrix of names, a total of $O(n^2 \log n)$ time and space. Suffix trees of each column in each of M_r^j, $1 \leq j \leq \log n$, are constructed in linear $O(n^2)$ time and space for each column, overall $O(n^2 \log n)$, and the preprocessing of Amir et al. [2] for weighted ancestor queries is also linear in time and space. In total, the complexity of preprocessing is $O(n^2 \log n)$ time and space.

5.1.1 Queries Used in Algorithm

Once the preprocessing is performed, we can make use of three kinds of efficient queries later on in our algorithm.

Query 1: Vertical Periodicity Query. Given an h-periodic submatrix S within matrix M, what is the vertical period of S?

A Vertical Periodicity Query can be answered in constant time by a SPQ in a column of one of the matrices of names $M_r^j, 1 \leq j \leq \log n$. If S has width 2^j, we use M_r^j. Suppose S begins in row α and ends in row β of M. We ask a SPQ in the column of M_r^j in which S begins with indices α and β that indicate the starting and ending rows of S in M. If S has width c such that $2^j < c < 2^{j+1}$, it is sufficient to ask a SPQ on the first 2^j columns of S. Since S is h-periodic, all of its remaining columns must appear in the first 2^j columns, and they do not affect the vertical periodicity of S. For example, in Figure 1, the Vertical Periodicity Query (on 2 columns) will answer 4 for the 8x3 highlighted submatrix at position (3, 14) and the Vertical Periodicity Query will answer 7 for the 14x3 highlighted submatrix at position (3, 14).

Query 2: Vertical Extension Query. Given a submatrix R that is a 2D repetition of height r, and an integer $x < r$, can R be extended vertically by x rows?

A Vertical Extension Query can be answered in $O(1)$ time as follows. We can compute the v-period v of R using Query 1. Let R^a be the submatrix R extended above by x rows. We do not know if R^a is h-periodic so we do not use Query 1. Let the width c of R satisfy $2^j \leq c < 2^{j+1}$. To compute the v-periodic of R^a, we use two SPQs in $O(1)$ time: one query for a prefix of size 2^j and another query for a suffix of size 2^j in a column of M_r^j. If both answers to the SPQs are equivalent to v, then the answer to the Vertical Extension Query is *yes*, meaning that the repetition R can be extended above by x rows. Otherwise, we perform the same computation for R^b, the submatrix R extended below by x rows. If the answers to both SPQs for R^b are equivalent to v, the answer to the Vertical Extension Query is *yes*. Otherwise, the answer to the vertical extension query is *no*, meaning that the repetition R cannot be extended by x rows up or down. For example, in Figure 1, a Vertical Extension Query by 6 rows on the 8x3 2D repetition beginning at position (3, 14) answers *no* even though it results in a 2D repetition since the vertical period grows with the vertical extension.

Query 3: Vertical Squares Query. Given a column c in M_r^j, $1 \leq j \leq \log n$, a position $1 \leq p \leq n$ within the column, and $1 \leq i \leq \log n$, locate each vertical square beginning at position (c, p) with height $2^i < r < 2^{i+1}$.

We use the data structure of the Vertical Squares Preprocessing described in Step 3 of the preprocessing. To answer the query we ask an $O(\log \log n)$ time weighted ancestor query on suffix p of column c in M_r^j with weight $2^{i+1} - 1$. The returned node's link to the closest marked ancestor yields such a square if one exists. Later, in Lemma 20 we prove that there are at most two answers to this query, hence, one additional link may need to be followed.

5.2 Step 2: Populate the Set $\mathbb{H}$

In this section, we find h-periodic submatrices of height 2^i in the input matrix. A 1D search, e.g. [16, 6], for runs across each row in M_c^i yields a set of h-periodic submatrices. These submatrices are necessarily maximal in their widths but not their heights since the height is fixed at 2^i for some i. Since a 1D row of length n can contain $O(n)$ repetitions [16], each row in M_c^i can contain $O(n)$ h-periodic submatrices. Thus, each matrix of names can contain $O(n^2)$ h-periodic submatrices, yielding a total of $O(n^2 \log n)$ h-periodic submatrices over the $\log n$ matrices of names. These submatrices may or may not be v-periodic. However, we will use them as a starting point for our search.

▶ **Definition 12.** Let $\mathbb{H}$ denote the set of all horizontally maximal h-periodic submatrices of height 2^i, for all $1 \leq i \leq \log n$.

▶ **Lemma 13.** *The procedure described in the previous paragraph finds every horizontally maximal h-periodic submatrix with height 2^i, for each $1 \leq i \leq \log n$, in input matrix M, i.e. we can find the complete set $\mathbb{H}$, in $O(n^2 \log n)$ time.*

Proof. Let I be a horizontally maximal h-periodic submatrix of height 2^i, $1 \leq i \leq \log n$, in M. There must be a subrow of M_c^i that corresponds exactly to I. By the correctness of the 1D search algorithm for runs across the rows of M_c^i, I will be found as a maximal 1D run. The algorithms of [16, 6] run in linear time on each of the $O(n)$ rows of length n in the $O(\log n)$ texts, resulting in $O(n^2 \log n)$ time overall. ◀

In Step 3 and and Step 4 we use the set $\mathbb{H}$ as the starting point for our search for all 2D maximal repetitions. We prove that each 2D maximal repetition in the desired output has a representative in the set $\mathbb{H}$ that shares its h-period. Thus, our algorithm processes each element in $\mathbb{H}$, extending it possibly in several ways, yielding different size repetitions.

▶ **Lemma 14.** *Each maximal 2D repetition R of height $2^i \leq r < 2^{i+1}, 1 \leq i \leq \log n$, has a representative $R' \in \mathbb{H}$ that overlaps R by 2^i rows and shares a corner on the left with R. R also has a representative $R'' \in \mathbb{H}$ that overlaps R by 2^i rows and shares a corner on the right with R.*

Proof. Let R be a maximal 2D repetition of height r. If $r = 2^i, 1 \leq i \leq \log n$, then $R \in \mathbb{H}$ and R is its own representative. Now suppose $2^i < r < 2^{i+1}$. Let $\hat{R}$ denote the prefix 2^i rows of R. Let $\check{R}$ denote the suffix 2^i rows of R. If either $\hat{R} \in \mathbb{H}$ or $\check{R} \in \mathbb{H}$ then a corner on each side is shared with a member of $\mathbb{H}$ and we do not need to consider other scenarios.

Now suppose $\hat{R} \notin \mathbb{H}$ and $\check{R} \notin \mathbb{H}$. This implies that both $\hat{R}$ and $\check{R}$ are not horizontally maximal. Suppose both $\hat{R}$ and $\check{R}$ need to be extended on the left to attain horizontal maximality. This implies that R needs to be extended on the left to attain horizontal maximality. This contradicts the fact that R is a maximal 2D repetition. Thus, R shares either its upper or lower left corner with a member of $\mathbb{H}$. The same argument can be used for the existence of a representative in $\mathbb{H}$ that shares a right corner with R. ◀

▶ **Corollary 15.** *Each maximal 2D repetition R shares either its upper left corner or its lower left corner with some element of $\mathbb{H}$.*

▶ **Corollary 16.** *Let $\mathbb{A}$ be the set of all horizontal prefixes in $\mathbb{H}$. Every maximal 2D repetition in M is the result of a vertical extension on some element of $\mathbb{A}$.*

▶ **Lemma 17.** *Let R be a maximal 2D repetition of height $2^i \leq r < 2^{i+1}, 1 \leq i \leq \log n$, with representatives $R' \in \mathbb{H}$ and $R'' \in \mathbb{H}$. R' and R'' both have the same h-period as R.*

Proof. Let h be the h-period of R. Let h' be the h-period of R'. Suppose $h' > h$. This is impossible since R cannot include fewer rows than R'. Suppose $h' < h$. This means that the Least Common Multiple (LCM) of the periods of the rows in R is larger than the LCM of the periods of the rows in R'. This is only possible if some row in R that is not part of R' has a period larger than that of any row in R'. This implies that some row in R does not occur in R'. This is impossible since we know that more than half of the vertical repetition in R occurs in R', and all the rows of R must occur in R' as well. Thus $h' = h$. The same argument can be made for the h-period of R''. ◀

Following Corollary 16, our algorithm will iterate through the elements in $\mathbb{H}$ and attempt to extend them downward and upward[5]. Since we know that the repetitions in $\mathbb{H}$ are maximal in width, it is not necessary to check horizontal maximality. However, it is possible that by reducing the width, an element in $\mathbb{H}$ can be extended to a taller height. (See the three 2D maximal repetitions beginning at position (13, 2) in Figure 1.) Hence, the task of extension is non-trivial. It is further complicated by the fact that we do not know the v-period of the elements in $\mathbb{H}$. In fact, some elements in $\mathbb{H}$ may not be v-periodic and yet may possibly be extendable into v-periodic matrices. (See the 14x3 2D maximal repetition at position (3, 14) and the 11x4 2D maximal repetition at (3, 3) in Figure 1.) Thus, we consider these two cases separately in the following two subsections. In Section 5.3, we consider the representatives in $\mathbb{H}$ that are v-periodic and identify all 2D maximal repetitions of height $2^i \leq r < 2^{i+1}$ whose prefix 2^i rows are v-periodic. Then, in Section 5.4, we locate the maximal 2D repetitions of height $2^i < r < 2^{i+1}$ whose prefix 2^i rows do not contain two complete copies of their v-periods.

5.3 Step 3: Extending 2D Repetitions Vertically

We begin by performing a Vertical Periodicity Query on each element in $\mathbb{H}$. If the element is v-periodic then it is processed in Step 3.

[5] The rest of the paper discusses the downward direction, the upward direction is analogous.

Algorithm 1 Find Maximal Height.

```
Input: 2D repetition R of height $2^i \leq r < 2^{i+1}$
Output: maximal height $r$ with width of R
                                        ▷ perform binary search to extend R
$j \leftarrow i - 1$                          ▷ we will try to extend by $2^j$ rows
while $j > -1$ do                           ▷ last extension should be by 1 row
    if Vertical Extension Query(R,$2^j$) then
        $r \leftarrow r + 2^j$                        ▷ R is extended by $2^j$ rows
    end if
    $j \leftarrow j - 1$         ▷ decrementing $j$ by 1 in effect halves the size of the extension
end while
```

Let R' denote an element in $\mathbb{H}$ that is both h-periodic and v-periodic. Algorithm 1 attempts to add rows to R' while maintaining its full width. A binary search procedure performs this extension by performing a sequence of Vertical Extension Queries. Once we have determined the maximal height for the full width, it is necessary to attempt to extend narrower widths. The following lemma shows that when there are several 2D maximal repetitions with the same primitive root that share a corner we get a progression of increasing heights and corresponding decreasing widths. For example, position (13, 2) of Figure 1 depicts several 2D repetitions with the same primitive root all starting at the same position.

▶ **Lemma 18.** *Let R be a maximal 2D repetition beginning at position (i, j) with dimensions $r_1 \times r_2$ and primitive root w of size $w_1 \times w_2$. For any other maximal 2D repetition R' beginning at (i, j) with dimensions $r_1' \times r_2'$ and the same primitive root w, $r_1' > r_1$ if and only if $r_2' < r_2$.*

Proof. The proof follows from the definition of maximality. ◀

This monotonicity property gives us the ability to use the modified binary search presented in Algorithm 1 on the potential heights of a 2D repetition. To illustrate Algorithm 1, we can consider the 13x4 maximal 2D repetition at position (2, 10) in Figure 1. We begin with the representative in $\mathbb{H}$ which has $2^3 = 8$ rows. We first try to extend downwards by 4 rows and succeed. Then we try to extend downwards by 2 more rows and fail. Finally, we try to extend downwards by 1 additional row and succeed, resulting in a maximal 2D repetition of height 13.

Algorithm 2 is the outer loop; its job is to compute the widths for which it needs Algorithm 1 to compute the corresponding maximal heights. For each repetition R of height $2^i < r < 2^{i+1}$, Algorithm 2 first checks whether it can be extended to the full height of 2^{i+1}. If it can, then this particular output will be found in another iteration, and the representative has been completed being processed. Otherwise, Algorithm 1 is called. Let r' denote the maximal height returned by Algorithm 1 for some repetition R. The full width of R cannot be extended even one row past r' since the last Vertical Extension Query failed at the end of Algorithm 1. Hence, a Longest Common Prefix (LCP) query in M between the substring of row $r' + 1$ directly below R and the row that is a v-period above it will determine the next width to extend. If the LCP suffices to admit two horizontal copies of the primitive root, we attempt to extend further downwards with Algorithm 1, passing a 2D repetition whose width is the answer to the LCP query and whose height is $r' + 1$. This process continues until either the width is too narrow or the height becomes 2^{i+1}.

Algorithm 2 Find Maximal Repetitions with v-periodic prefix 2^i rows.

```
Input: set $\mathbb{H}$ of h-periodic submatrices, each with height $2^i$
Output: maximal 2D repetitions with height $2^i \le r \le 2^{i+1}$ that are v-periodic in their prefix $2^i$ rows
for all $R \in \mathbb{H}$ do                ▷ $R$ has height $r = 2^i$, width $c$, and h-period $h$
    Ask Vertical Periodicity Query on $R$ to get vertical period $v$ of $R$
    if $v \le \frac{r}{2}$ then                                   ▷ $R$ is v-periodic
        repeat
            $d \leftarrow 2^{i+1} - r$                    ▷ $d$ is distance to height $2^{i+1}$
            if Vertical Extension Query($R$, d) then
                break              ▷ Don't process $R$ if it extends to height $2^{i+1}$.
            end if
            Call Algorithm 1 to compute maximal height $r'$ for $R$
            $r \leftarrow r'$
            Output $R$ with new height
                      ▷ see if can extend $R$ further downwards by decreasing its width
            $\ell \leftarrow$ LCP between row $r - v + 1$ in $R$ and row $r + 1$ below $R$
            if $\ell \ge 2h$ then
                $c \leftarrow \ell$
                $r \leftarrow r + 1$
            end if
        until $\ell < 2h$ or $r \ge 2^{i+1}$
              ▷ continue looking for taller and narrower 2D repetitions as long as width is h-periodic and height is less than $2^{i+1}$
    end if
end for
```

5.4 Step 4: Unknown Vertical Period

In this section we process *all elements* of $\mathbb{H}$ to discover v-periods that were previously unknown. This can happen in one of two ways in a 2D repetition of height $2^i < r < 2^{i+1}$, $1 \le i \le \log n$.

1. The first 2^i rows of a repetition are not v-periodic. For example, `abaababa` with $i = 2$, and each character of the string is a metacharacter representing a subrow in M. (See the 11x4 maximal repetition at position (3, 3) in Figure 1.)
2. The first 2^i rows are v-periodic, but there is another v-period that comes into existence when rows are added. For example, `aabaaabaabaaab` with $i = 3$ and each character of the string is a metacharacter representing a subrow in M. In the first 2^i rows of this submatrix, we have the periodic string `aabaaaba` with period `aaba`. The first 14 rows are also periodic with period `aabaaab` of size 7. (See the two maximal repetitions at position (3, 14) in Figure 1.)

The following two lemmas identify the key characteristics of a 2D repetition of height $2^i < r < 2^{i+1}$, $1 \le i \le \log n$, and v-period v such that v is not a vertical period in the first 2^i rows of R, i.e., $v > 2^{i-1}$.

▶ **Lemma 19.** *In a 2D repetition R of height $2^i < r < 2^{i+1}$ whose 2^i prefix rows are not v-periodic, the exponent e of the v-period v is between 2 and 4, i.e., $2 \le e < 4$.*

Proof. Let $|v|$ be the size of v. We know $|v| > \frac{1}{2}2^i = 2^{i-1}$ since the first 2^i rows of R are not periodic in v. Thus, $|v| \geq 2^{i-1} + 1$. We know that the height of R is at most $2^{i+1} - 1$. The longest possible height of R divided by its shortest possible root yields the largest possible exponent for the v-period.

$$e \leq \frac{2^{i+1} - 1}{2^{i-1} + 1} < \frac{2^{i+1}}{2^{i-1}} = 4.$$

By the definition of an exponent for a period, $e \geq 2$. Overall, $2 \leq e < 4$. ◀

▶ **Lemma 20.** *Let I be an h-periodic matrix of height $2^i < r < 2^{i+1}$. No more than 2 v-periods v can occur at the beginning of I such that the first 2^i rows of I are not periodic in v.*

Proof. Suppose v_1 and v_2 are the smallest v-periods in I such that $v_1 > 2^{i-1}$ and $v_2 > 2^{i-1}$. Suppose I has a third v-period v_3 in which the first 2^i rows are not periodic in v_3. Then $v_3 \geq v_1 + v_2$ [10]. Thus, $v_3 > 2^i$. This implies that v_3 cannot occur twice in I of height $r < 2^{i+1}$ and I cannot be v-periodic in v_3. Thus, a third v-period of height larger than 2^{i-1} cannot exist in I. ◀

By Lemmas 19 and 20, we know that we are looking for 2D repetitions that contain either 2 or 3 complete copies of their v-periods and that each element of $\mathbb{H}$ will extend to at most 2 new v-periods. Hence, we are looking for at most two squares that begin with I and have height $2^i < r < 2^{i+1}$. Suppose I has width c such that $2^j \leq c < 2^{j+1}$ and that I begins in row α. We ask a Vertical Squares Query on the column at which I begins in M_r^j, α, and i. Once we identify the 1 or 2 v-periods, if they occur, we revert back to Step 3 of the algorithm (when the v-period is known) and use the procedure described in Algorithm 2 to find the set of maximal 2D repetitions corresponding to each v-period we have identified.

5.5 Algorithm Correctness and Time Complexity

▶ **Theorem 21.** *Let M be an $n \times n$ matrix. Our algorithm finds all maximal 2D repetitions that occur in M.*

Proof. By Lemma 14 every repetition in the output has a representative in $\mathbb{H}$. It remains to show that we hit upon every repetition R in the output with some vertical extension of an element $R' \in \mathbb{H}$ that is a prefix or suffix of R. By Corollary 15, we know that R' shares a corner on the left with R. If R and R' have the same primitive root (Step 3), then by Lemma 18 the successive binary searches will hit upon every output. Now suppose that R' has a different primitive root than R. R' and R must have the same h-period, by Lemma 17, so R' and R must have different v-periods. By Lemma 20, there can be no more than two possible v-periods to try extending with a binary search. One of the extensions must be R, by Corollary 16. Hence, our algorithm identifies all maximal 2D repetitions in M. ◀

▶ **Theorem 22.** *Let M be an $n \times n$ matrix. Our algorithm finds all maximal 2D repetitions in M in $O(n^2 \log n \log\log n + \rho \log n)$ time, where ρ is the number of maximal 2D repetitions that occur in M.*

Proof. Step 1 of the Algorithm Outline, the preprocessing, was shown in Section 5.1 to be done in $O(n^2 \log n)$ time. For Step 2, a linear time 1D search (e.g., [16, 6]) for runs across each row in M_c^i yields the set $\mathbb{H}$ in $O(n^2 \log n)$ time and space.

In Step 3, we iterate through the $O(n^2 \log n)$ elements in $\mathbb{H}$ and perform a constant time Vertical Periodicity Query on each element. Then, Algorithm 2 is called on each v-periodic element. Algorithm 2 performs a Vertical Extension Query and LCP query in constant time. It also calls Algorithm 1 for each representative for each width that is necessary to check. Algorithm 1 runs in $O(i) = O(\log n)$ time. However, the only widths that are checked are those that will certainly produce output. Therefore, we can charge the time spent running Algorithm 1 to the output it generates, yielding $O(\rho \log n)$ where ρ equals the number of repetitions reported. Each repetition is reported at most twice since it can be found once by the upward and downward extensions. For Step 4, we find the 1 or 2 v-periods of interest in $O(\log \log n)$ time using the Vertical Squares Query and again use Algorithm 2 to find the corresponding maximal 2D repetitions in $O(n^2 \log n + \rho \log n)$ time. Hence, the total time complexity of our algorithm is $O(n^2 \log n \log \log n + \rho \log n)$. ◀

References

1 A. Amir and G. Benson. Two-dimensional periodicity in rectangular arrays. *SIAM J. Comput.*, 27(1):90–106, 1998. doi:10.1137/S0097539795298321.

2 A. Amir, G. M. Landau, M. Lewenstein, and D. Sokol. Dynamic text and static pattern matching. *ACM Trans. Algorithms*, 3(2):19, 2007. doi:10.1145/1240233.1240242.

3 A. Apostolico and V. E. Brimkov. Fibonacci arrays and their two-dimensional repetitions. *Theor. Comput. Sci.*, 237(1-2):263–273, 2000. doi:10.1016/S0304-3975(98)00182-0.

4 A. Apostolico and V. E. Brimkov. Optimal discovery of repetitions in 2d. *Discrete Applied Mathematics*, 151(1-3):5–20, 2005. doi:10.1016/j.dam.2005.02.019.

5 N. Bacquey. Primitive roots of bi-periodic infinite pictures. In *Words 2015*, 2015.

6 H. Bannai, T. I, S. Inenaga, Y. Nakashima, M. Takeda, and K. Tsuruta. A new characterization of maximal repetitions by Lyndon trees. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 562–571, 2015. doi:10.1137/1.9781611973730.38.

7 H. Bannai, T. I, S. Inenaga, Y. Nakashima, M. Takeda, and K. Tsuruta. The "runs" theorem. *SIAM J. Comput.*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.

8 M. Crochemore, L. Gasieniec, R. Hariharan, S. Muthukrishnan, and W. Rytter. A constant time optimal parallel algorithm for two-dimensional pattern matching. *SIAM J. Comput.*, 27(3):668–681, 1998. doi:10.1137/S0097539795280068.

9 M. Crochemore, L. Ilie, and W. Rytter. Repetitions in strings: Algorithms and combinatorics. *Theor. Comput. Sci.*, 410(50):5227–5235, 2009. doi:10.1016/j.tcs.2009.08.024.

10 M. Crochemore and W. Rytter. Sqares, cubes, and time-space efficient string searching. *Algorithmica*, 13(5):405–425, 1995. doi:10.1007/BF01190846.

11 G. Gamard and G. Richomme. Coverability in two dimensions. In A. H. Dediu, E. Formenti, C. Martín-Vide, and B. Truthe, editors, *Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings*, volume 8977 of *Lecture Notes in Computer Science*, pages 402–413. Springer, 2015. doi:10.1007/978-3-319-15579-1_31.

12 G. Gamard, G. Richomme, J. Shallit, and T. J. Smith. Periodicity in rectangular arrays. *Inf. Process. Lett.*, 118:58–63, 2017. doi:10.1016/j.ipl.2016.09.011.

13 D. Gusfield and J. Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *J. Comput. Syst. Sci.*, 69(4):525–546, 2004. doi:10.1016/j.jcss.2004.03.004.

14 R. M. Karp, R. E. Miller, and A. L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In *Proceedings of the 4th Annual ACM Symposium on Theory of Computing, May 1-3, 1972, Denver, Colorado, USA*, pages 125–136, 1972. doi:10.1145/800152.804905.

15 T. Kociumaka, J. Radoszewski, W. Rytter, and T. Walen. Internal pattern matching queries in a text and applications. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 532–551, 2015. doi:10.1137/1.9781611973730.36.

16 R. M. Kolpakov and G. Kucherov. Finding maximal repetitions in a word in linear time. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 596–604. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814634.

17 S. Marcus and D. Sokol. 2d Lyndon words and applications. *Algorithmica*, 77(1):116–133, 2017. doi:10.1007/s00453-015-0065-z.

18 E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995. doi:10.1007/BF01206331.

Approximate Convex Intersection Detection with Applications to Width and Minkowski Sums

Sunil Arya[1]
Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong
arya@cse.ust.hk

Guilherme D. da Fonseca[2]
Université Clermont Auvergne, LIMOS, and INRIA Sophia Antipolis, France
fonseca@isima.fr

David M. Mount[3]
Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA
mount@cs.umd.edu

Abstract

Approximation problems involving a single convex body in $\mathbb{R}^d$ have received a great deal of attention in the computational geometry community. In contrast, works involving multiple convex bodies are generally limited to dimensions $d \leq 3$ and/or do not consider approximation. In this paper, we consider approximations to two natural problems involving multiple convex bodies: detecting whether two polytopes intersect and computing their Minkowski sum. Given an approximation parameter $\varepsilon > 0$, we show how to independently preprocess two polytopes $A, B \subset \mathbb{R}^d$ into data structures of size $O(1/\varepsilon^{(d-1)/2})$ such that we can answer in polylogarithmic time whether A and B intersect approximately. More generally, we can answer this for the images of A and B under affine transformations. Next, we show how to ε-approximate the Minkowski sum of two given polytopes defined as the intersection of n halfspaces in $O(n \log(1/\varepsilon) + 1/\varepsilon^{(d-1)/2+\alpha})$ time, for any constant $\alpha > 0$. Finally, we present a surprising impact of these results to a well studied problem that considers a single convex body. We show how to ε-approximate the width of a set of n points in $O(n \log(1/\varepsilon) + 1/\varepsilon^{(d-1)/2+\alpha})$ time, for any constant $\alpha > 0$, a major improvement over the previous bound of roughly $O(n + 1/\varepsilon^{d-1})$ time.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Minkowski sum, convex intersection, width, approximation

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.3

1 Introduction

Approximation problems involving a single convex body in d-dimensional space have received a great deal of attention in the computational geometry community [4, 9, 10, 11, 12, 18, 19, 45]. Recent results include near-optimal algorithms for approximating the convex hull of a set of points [9, 19], as well as an optimal data structure for answering approximate polytope

[1] Research supported by the Research Grants Council of Hong Kong, China under project number 16200014.
[2] Research supported by the European Research Council under ERC Grant Agreement number 339025 GUDHI (Algorithmic Foundations of Geometric Understanding in Higher Dimensions).
[3] Research supported by NSF grant CCF–1618866.

© Sunil Arya, Guilherme D. da Fonseca, and David M. Mount;
licensed under Creative Commons License CC-BY

membership queries [11]. In contrast, works involving multiple convex bodies are generally limited to dimensions $d \leq 3$ and/or do not consider approximation [2, 13, 29, 30, 44]. In this paper we present new approximation algorithms to natural problems that either involve multiple convex polytopes or result from such an analysis:

- Determining whether two convex polytopes A and B intersect
- Computing the Minkowski sum, $A \oplus B$, of two convex polytopes
- Computing the width of a convex polytope A (which results from an analysis of the Minkowski sum $A \oplus (-A)$)

Throughout we assume that the input polytopes reside in $\mathbb{R}^d$ and are full-dimensional, where the dimension d is a fixed constant. Polytopes may be represented either as the convex hull of n points (*point representation*) or as the intersection of n halfspaces (*halfspace representation*). In either case, n denotes the *size* of the polytope.

1.1 Convex Intersection

Detecting whether two geometric objects intersect and computing the region of intersection are fundamental problems in computational geometry. Geometric intersection problems arise naturally in a number of applications. Examples include geometric packing and covering, wire and component layout in VLSI, map overlay in geographic information systems, motion planning, and collision detection. Several surveys present the topics of collision detection and geometric intersection [33, 36, 37].

The special case of detecting the intersection of convex objects has received a lot of attention in computational geometry. The static version of the problem has been considered in $\mathbb{R}^2$ [39, 42] and $\mathbb{R}^3$ [20, 38]. The data structure version where each convex object is preprocessed independently has been considered in $\mathbb{R}^2$ [13, 21, 22, 25] and $\mathbb{R}^3$ [13, 22, 25, 26].

Recently, Barba and Langerman [13] considered the problem in higher dimension. They showed how to preprocess convex polytopes in $\mathbb{R}^d$ so that given two such polytopes that have been subject to affine transformations, it can be determined whether they intersect each other in logarithmic time. However, the preprocessing time and storage grow as the combinatorial complexity of the polytope raised to the power $\lfloor d/2 \rfloor$. Since the combinatorial complexity of a polytope with n vertices can be as high as $\Theta(n^{\lfloor d/2 \rfloor})$, the storage upper bound is roughly $O(n^{d^2/4})$. This high complexity motivates the study of approximations to the problem.

We define approximation in a manner that is sensitive to direction. Consider any convex body K in $\mathbb{R}^d$ and any $\varepsilon > 0$. Given a nonzero vector $v \in \mathbb{R}^d$, define $\Pi_v(K)$ to be the minimum slab defined by two hyperplanes that enclose K and are orthogonal to v. Define the *directional width* of K with respect to v, $\mathrm{width}_v(K)$, to be the perpendicular distance between these hyperplanes. Let $\Pi_{v,\varepsilon}(K)$ be the central expansion of $\Pi_v(K)$ by a factor of $1+\varepsilon$, and define K_ε to be the intersection of these expanded slabs over all unit vectors v. It can be shown that for any v, $\mathrm{width}_v(K_\varepsilon) = (1+\varepsilon)\,\mathrm{width}_v(K)$. An *$\varepsilon$-approximation* of K is any set K' (which need not be convex) such that $K \subseteq K' \subseteq K_\varepsilon$. This defines an *outer* approximation. It is also possible to define an analogous notion of *inner* approximation in which each directional width is no smaller than $1-\varepsilon$ times the true width. Our results can be extended to either type of approximation.

Given a discrete point set S in $\mathbb{R}^d$, an *ε-kernel* of S is any subset $Q \subseteq S$ such that $\mathrm{conv}(Q)$ is an inner ε-approximation of $\mathrm{conv}(S)$ [4]. It is well known that $O(1/\varepsilon^{(d-1)/2})$ points are sufficient and sometimes necessary in an ε-kernel. Kernels efficiently approximate the convex hull and as such have been used to obtain fast approximation algorithms to several problems such as diameter, minimum width, convex hull volume, minimum enclosing cylinder, minimum enclosing annulus, and minimum-width cylindrical shell [4, 5].

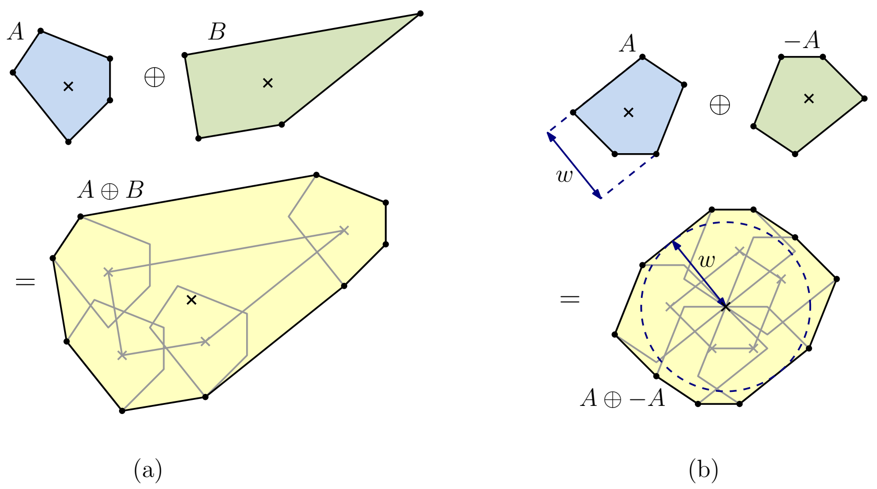


Figure 1 Minkowski sum and its relationship to width.

In the ε-approximate version of convex intersection, we are given two convex bodies A and B and a parameter $\varepsilon > 0$. If $A \cap B \neq \emptyset$, then the answer is "*yes*." If $A_\varepsilon \cap B_\varepsilon = \emptyset$, then the answer is "*no*." Otherwise, either answer is acceptable. The *ε-approximate polytope intersection problem* is defined as follows. A collection of two or more convex polytopes in $\mathbb{R}^d$ are individually preprocessed (with knowledge of ε). Given any two preprocessed polytopes, A and B, the query determines whether A and B intersect approximately. In general, the query algorithm can be applied to any affine transformation of the preprocessed polytopes.

▶ **Theorem 1.** *Given a parameter $\varepsilon > 0$ and two polytopes $A, B \subset \mathbb{R}^d$ each of size n (given either using a point or halfspace representation), we can independently preprocess each polytope into a data structure in order to answer ε-approximate polytope intersection queries with query time $O(\mathrm{polylog}\,\frac{1}{\varepsilon})$, storage $O(1/\varepsilon^{(d-1)/2})$, and preprocessing time $O(n \log \frac{1}{\varepsilon} + 1/\varepsilon^{(d-1)/2+\alpha})$, where α is an arbitrarily small positive constant.*

The space is nearly optimal because there is a lower bound of $\Omega(1/\varepsilon^{(d-1)/2})$ on the worst-case bit complexity of representing an ε-approximation of a polytope [11].

1.2 Minkowski Sum

Given two convex bodies $A, B \subset \mathbb{R}^d$, the *Minkowski sum* $A \oplus B$ is defined as $\{p + q : p \in A,\ q \in B\}$ (see Figure 1(a)). Minkowski sums have found numerous applications in motion planning [7, 31], computer-aided design [44], computational biology [40], satellite layout [15], and image processing [35]. Minkowski sums have been also been well studied in the context of discrete and computational geometry [1, 3, 29, 32, 43].

It is well known that in dimension $d \geq 3$, the number of vertices in the Minkowski sum of two polytopes can grow as rapidly as the product of the number of vertices in the two polytopes [7]. This has led to the study of algorithms to compute approximations to Minkowski sums in $\mathbb{R}^3$ [2, 30, 44]. In this paper, we show how to approximate the Minkowski sum of two convex polytopes in $\mathbb{R}^d$ in near-optimal time.

▶ **Theorem 2.** *Given a parameter $\varepsilon > 0$ and two polytopes $A, B \subset \mathbb{R}^d$ each of size n (given either using a point or halfspace representation), it is possible to construct an ε-approximation of $A \oplus B$ of size $O(1/\varepsilon^{(d-1)/2})$ in $O(n \log \frac{1}{\varepsilon} + 1/\varepsilon^{(d-1)/2+\alpha})$ time, where α is an arbitrarily small positive constant.*

The output representation can be either point-based or halfspace-based.

1.3 Width

Define the directional width of a set S of n points to be the directional width of $\mathrm{conv}(S)$. The *width* of S is the *minimum* over all directional widths. The *maximum* over all directional widths is equal to the diameter of S. Both problems can be approximated using the ε-kernel of S. After successive improvements [4, 6, 8, 14, 18], algorithms to compute ε-kernels and to ε-approximate the diameter in roughly $O(n+1/\varepsilon^{d/2})$ time have been independently discovered by Chan [19] and the authors [9]. Somewhat surprisingly, these works offer no improvement to the running time to approximate the width [4, 17, 18, 28, 45], which Chan [19] posed as an open problem. The fastest known algorithms date from over a decade ago and take roughly $O(n+1/\varepsilon^{d-1})$ time [17, 18].

Agarwal *et al.* [2] showed that the width of a convex body K is equal to the minimum distance from the origin to the boundary of the convex body $K \oplus (-K)$ (see Figure 1(b)). Using Theorem 2, we can approximate the width by computing an ε-approximation of $K \oplus (-K)$ represented as the intersection of halfspaces and then determining the closest point to the origin among all bounding hyperplanes. The following presents this result.

▶ **Theorem 3.** *Given a set S of n points in $\mathbb{R}^d$ and an approximation parameter $\varepsilon > 0$, it is possible to compute an ε-approximation to the width of S in $O(n \log \frac{1}{\varepsilon} + 1/\varepsilon^{(d-1)/2+\alpha})$ time, where α is an arbitrarily small positive constant.*

1.4 Techniques

Our algorithms and data structure are based on a data structure defined by a hierarchy of Macbeath regions [9, 11], which answers approximate directional width queries in polylogarithmic time. First, we show how to use this data structure as a black box to answer approximate polytope intersection queries by transforming the problem to a dual setting and performing a multidimensional convex minimization. Next, we show how to use approximate polytope intersection queries to compute ε-approximations of the Minkowski sum. The approximation to the width follows directly.

Since we only access the input polytopes through a data structure for approximate directional width queries, our results apply in much more general settings. For example, we could answer in polylogarithmic time whether the Minkowski sum of two polytopes (preprocessed independently) approximately intersects a third polytope. Our techniques are also amenable to other polytope operations such as intersection and convex hull of the union, as long as the model of approximation is defined accordingly.

The preprocessing time of the approximate directional width data structure we use is $O(n \log \frac{1}{\varepsilon} + 1/\varepsilon^{(d-1)/2+\alpha})$, for arbitrarily small $\alpha > 0$. If this preprocessing time is reduced in the future, the complexity of our algorithms becomes equal to the preprocessing time plus $O((1/\varepsilon^{(d-1)/2}) \operatorname{polylog} \frac{1}{\varepsilon})$.

2 Preliminaries

In this section we present a number of results, which will be used throughout the paper. The first provides three basic properties of Minkowski sums. The proof can be found in standard sources on Minkowski sums (see, e.g., [41]).

▶ **Lemma 4.** *Let $A, B \subset \mathbb{R}^d$ be two (possibly infinite) sets of points. Then:*

(a) *$A \cap B \neq \emptyset$ if and only if $O \in A \oplus (-B)$, where O is the origin.*

(b) $\mathrm{conv}(A \oplus B) = \mathrm{conv}(A) \oplus \mathrm{conv}(B)$.

(c) *For all nonzero vectors v, $\mathrm{width}_v(A \oplus B) = \mathrm{width}_v(A) + \mathrm{width}_v(B)$.*

Next, we recall a recent result of ours on answering directional width queries approximately [9], which we will use as a black box later in this paper. Given a set S of n points in a constant dimension d and an approximation parameter $\varepsilon > 0$, the answer to the *approximate directional width query* for a nonzero query vector v consists of a pair of points $p, q \in S$ such that $\mathrm{width}_v(\{p,q\}) \geq (1-\varepsilon)\ \mathrm{width}_v(S)$.

▶ **Lemma 5.** *Given a set S of n points in $\mathbb{R}^d$ and an approximation parameter $\varepsilon > 0$, there is a data structure that can answer ε-approximate directional width queries with query time $O(\log^2 \frac{1}{\varepsilon})$, space $O(1/\varepsilon^{(d-1)/2})$, and preprocessing time $O(n \log \frac{1}{\varepsilon} + 1/\varepsilon^{(d-1)/2+\alpha})$.*

2.1 Fattening

Existing algorithms and data structures for convex approximation often assume that the bodies have been fattened through an appropriate affine transformation. In the context of multiple bodies, this is complicated by the fact that different fattening transformations may be needed for the two bodies or their Minkowski sum. In this section we explore this issue.

Consider a convex body K in d-dimensional space $\mathbb{R}^d$. Given a parameter $0 < \gamma \leq 1$, we say that K is *γ-fat* if there exist concentric Euclidean balls B and B', such that $B \subseteq K \subseteq B'$, and $\mathrm{radius}(B)/\mathrm{radius}(B') \geq \gamma$. We say that K is *fat* if it is γ-fat for a constant γ (possibly depending on d, but not on ε or K). For a centrally symmetric convex body C, the body obtained by scaling C about its center by a factor of λ is called the λ-expansion of C.

Let K be a convex body. We say that a convex body C is a *λ-sandwiching* body for K if C is centrally symmetric and $C \subseteq K \subseteq C'$, where C' is a λ-expansion of C. John [34] proved tight bounds for the constant λ of a λ-sandwiching ellipsoid. This ellipsoid is referred to as the *John ellipsoid.*

▶ **Lemma 6.** *For every convex body K in $\mathbb{R}^d$, there exists a d-sandwiching ellipsoid. Furthermore, if K is centrally symmetric, there exists a $\sqrt{d}$-sandwiching ellipsoid.*

It is an immediate consequence of this lemma that for any convex body K there exists an affine transformation T such that $T(K)$ is $(1/d)$-fat. Any affine transformation that maps the John ellipsoid into a Euclidean ball will do. The following lemma generalizes this to hyperrectangles (see also Barequet and Har-Peled [14]).

▶ **Lemma 7.** *For every convex body K in $\mathbb{R}^d$, there exists a $(d^{3/2})$-sandwiching hyperrectangle.*

Proof. Let E denote the d-sandwiching ellipsoid for K, described in Lemma 6. By elementary geometry, there exists a $\sqrt{d}$-sandwiching hyperrectangle R for E. We claim that R is a $(d^{3/2})$-sandwiching hyperrectangle for K. To prove this claim, observe that $R \subseteq E \subseteq R'$ and $E \subseteq K \subseteq E'$, where R' is the $\sqrt{d}$-expansion of R and E' is the d-expansion of E. Letting R'' denote the d-expansion of R', it is easy to see that $E' \subseteq R''$. It follows that $R \subseteq E \subseteq K \subseteq E' \subseteq R''$. Since R'' is the d-expansion of R' and R' is the $\sqrt{d}$-expansion of R, it follows that R'' is the $(d^{3/2})$-expansion of R. This completes the proof. ◀

Next, let us consider fattening in the context of multiple bodies. The next two lemmas follow from elementary geometry and properties of Minkowski sums.

▶ **Lemma 8.** *Let C_1 and C_2 be λ-sandwiching bodies for K_1 and K_2, respectively. Then $C_1 \oplus C_2$ is a λ-sandwiching body for $K_1 \oplus K_2$.*

▶ **Lemma 9.** *Let K be a convex body. Given a λ-sandwiching polytope for K of constant complexity, we can compute a γ-fattening affine transformation T for K in constant time, where $\gamma = 1/(\lambda\sqrt{d})$.*

We conclude by showing that we can maintain a small amount of auxiliary information for any collection of convex bodies in order to determine the fattening transformation for the Minkowski sum of any two members of this library. We refer to the data structure for approximate directional width queries from Lemma 5 together with the additional information to determine the fattening transformation as the *augmented data structure* for approximate directional width queries.

▶ **Lemma 10.** *Consider any finite collection of convex polytopes in $\mathbb{R}^d$, and let $\gamma = 1/d^2$. It is possible to store information of constant size with each polytope such that in constant time we can compute a γ-fattening affine transformation for the Minkowski sum of any two polytopes from the collection. This information can be computed in time proportional to the size of the input polytope.*

Proof. At preprocessing time, we store the λ-sandwiching hyperrectangles R_i for each K_i, where $\lambda = d^{3/2}$. By Lemma 7, such hyperrectangles exist and they can be computed in time proportional to the size of the input polytope [23].

Suppose we want to compute a γ-fattening affine transformation for $K_i' \oplus K_j'$, where K_i' and K_j' are the result of applying (possibly different) affine transformations to K_i and K_j, respectively. Let C_i' and C_j' be the polytopes of constant complexity obtained by applying the corresponding affine transformations to R_i and R_j, respectively. Clearly, C_i' and C_j' are λ-sandwiching polytopes for K_i' and K_j', respectively. Thus, by Lemma 8, $C_i' \oplus C_j'$ is a λ-sandwiching polytope for $K_i' \oplus K_j'$. Note that this polytope has constant complexity and can be computed in constant time. Applying Lemma 9, we can use this polytope to compute a γ-fattening affine transformation for $K_i' \oplus K_j'$ in constant time, where $\gamma = 1/(\lambda\sqrt{d}) = 1/d^2$. ◀

The previous lemma holds more generally even when each of the polytopes are subject to any non-singular affine transformation and to the Minkowski sum of a constant number of polytopes.

2.2 Projective Duality and Width

Our algorithm for approximating the directional width of a point set is based on a *projective dual transformation*, which maps points into hyperplanes and vice versa. Each primal point $p = (p_1, \ldots, p_d) \in S$ is mapped to the dual hyperplane $p^* : x_d = p_1 x_1 + \cdots + p_{d-1} x_{d-1} - p_d$. Each primal hyperplane is mapped to a dual point in the same manner. This dual transformation has several well-known properties [24]. For example, the points in the lower convex hull of S map to the hyperplanes in the upper envelope.

Let H be a set of n hyperplanes in $\mathbb{R}^d$. Given a point $r \in \mathbb{R}^{d-1}$, the *thickness* of H at r, denoted $\mathrm{thick}_r(H)$ is defined as follows. Given $r \in \mathbb{R}^{d-1}$ and $t \in \mathbb{R}$, let (r,t) denote the point in $\mathbb{R}^d$ resulting by concatenating r and t. For the sake of illustration, we think of the d-th coordinate axis as being the vertical axis. Let $r' = (r, t_1)$ and $r'' = (r, t_2)$. We define $\mathrm{thick}_r(H)$ as the maximum difference $t_2 - t_1$ for points r', r'' in the hyperplanes in H. In other words, the thickness is the vertical distance between the intersection of the vertical line defined by r with the upper and lower envelopes of H. The following relates width and thickness.

▶ **Lemma 11.** *Consider two points $p, q \in \mathbb{R}^d$ and a vector $v = (v_1, \ldots, v_{d-1}, -1)$. Let p^*, q^* denote the dual hyperplanes and $v_{1,d-1} = (v_1, \ldots, v_{d-1})$. We have*

$$\mathrm{thick}_{v_{1,d-1}}(\{p^*, q^*\}) = \|v\| \, \mathrm{width}_v(\{p, q\}).$$

Proof. Given vectors u and v, let $u \cdot v$ denote the standard inner product. Assume without loss of generality that $p \cdot v \geq q \cdot v$. Clearly, v is nonzero, so $\text{width}_v(\{p, q\}) = (p \cdot v - q \cdot v)/\|v\|$. Let $p = (p_1, \dots, p_d)$ and $q = (q_1, \dots, q_d)$. The dual hyperplanes are

$$p^* : x_d = p_1 x_1 + \cdots + p_{d-1} x_{d-1} - p_d \quad \text{and} \quad q^* : x_d = q_1 x_1 + \cdots + q_{d-1} x_{d-1} - q_d.$$

If we set $x_1, \dots, x_{d-1} = v_{1,d-1}$ we have $t_2 = (p_1, \dots, p_{d-1}) \cdot v_{1,d-1} - p_d$ and $t_1 = (q_1, \dots, q_{d-1}) \cdot v_{1,d-1} - q_d$. Therefore

$$\begin{aligned} \text{thick}_{v_{1,d-1}}(H) &= t_2 - t_1 \\ &= (p_1, \dots, p_{d-1}) \cdot v_{1,d-1} - p_d - ((q_1, \dots, q_{d-1}) \cdot v_{1,d-1} - q_d) \\ &= p \cdot v - q \cdot v \\ &= \|v\| \, \text{width}_v(\{p, q\}). \end{aligned}$$ ◀

3 Approximate Convex Intersection

In this section, we will prove Theorem 1 for the case when the input polytopes are represented by points. Assume that we are given two polytopes A and B in the point representation. The objective is to preprocess A and B individually such that we can efficiently answer approximate intersection queries for A and B (or more generally for affine transformations of A and B).

Given a convex body K, $\varepsilon > 0$, and a point p, an *ε-approximate polytope membership query* is defined as follows. If $p \in K$, the answer is "*yes*," if $p \notin K_\varepsilon$, the answer is "*no*," and otherwise, either answer is acceptable. Our strategy to answer approximate intersection queries is based on reducing them to approximate polytope membership queries. This reduction is presented in the following lemma, which is a straightforward generalization of Lemma 4(a) to an approximate context. The proof follows from standard algebraic properties of Minkowski sums and the observation that K_ε can be expressed as $K \oplus \frac{\varepsilon}{2}(K \oplus -K)$, and is omitted from this version.

▶ **Lemma 12.** *Let $A, B \subset \mathbb{R}^d$ be two polytopes and $\varepsilon > 0$. Determining the ε-approximate intersection of A and B is equivalent to determining the ε-approximate membership of $O \in A \oplus (-B)$.*

The previous lemma relates approximate polytope intersection with an approximate membership of the origin in a polytope (Figure 2(a)). Determining whether the origin lies within the convex hull of a set of points S is a classic problem in computational geometry, which can be solved by linear programming. However, we are interested in a faster approximate solution that does not compute S explicitly. We cannot afford to preprocess an approximate polytope membership data structure for $A \oplus (-B)$ for each pair A and B, since the number of such pairs is quadratic in the number of input polytopes. Instead, we preprocess each input polytope individually, and we show next how to efficiently answer approximate polytope membership queries for $A \oplus (-B)$ by using augmented data structures for approximate directional width queries for A and B as black boxes.

▶ **Lemma 13.** *Given augmented data structures for answering ε-approximate directional width queries for polytopes A and B, we can answer ε-approximate membership queries for $A \oplus (-B)$ using $O(\text{polylog} \frac{1}{\varepsilon})$ queries to these data structures.*

Proof. Without loss of generality, we may translate space so that the query point coincides with the origin O. Let $K = A \oplus (-B)$, and let S be K's vertex set. (Note that K and S are not explicitly computed.)

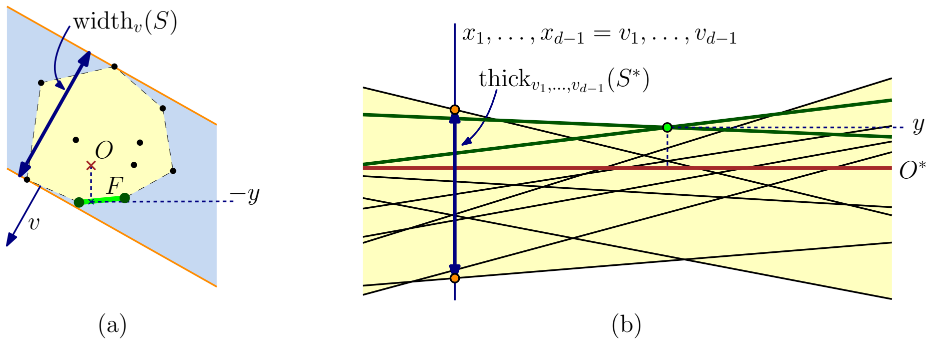

width$_v(S)$
O
F
$-y$
v
(a)
$x_1, \dots, x_{d-1} = v_1, \dots, v_{d-1}$
thick$_{v_1,\dots,v_{d-1}}(S^*)$
y
O^*
(b)

Figure 2 (a) Primal problem of determining if $O \in \text{conv}(S)$. (b) Dual problem of determining if the horizontal hyperplane O^* is between the upper and lower envelopes.

The problem of determining whether $O \in K$ is invariant to scaling and rotation about the origin. It will be helpful to perform some affine transformations that will guarantee certain properties for K. First, we apply Lemma 10 to fatten K and then apply a uniform scaling about the origin so that K's diameter is $\Theta(1)$. By fatness, K has a λ-sandwiching ball of radius $r = \Theta(1)$. If the origin either lies within the inner ball or outside the outer ball, then the answer is trivial. Otherwise, let $\Delta = 2\lambda r$ be the diameter of the outer ball. We may apply a rotation about the origin so that the center of this ball lies on the positive x_d axis at a point $(0, \dots, 0, \beta)$. Again, this scaling and rotation can be computed in constant time using the augmented information. It follows that the coordinates of the points of S have absolute values at most $\Delta = \Theta(1)$.

In summary, there exists an affine transformation computable in constant time such that after applying this transformation, the query point lies at the origin, $K = \text{conv}(S)$ is sandwiched between two concentric balls of constant radii centered at $c = (0, \dots, 0, \beta)$, where $0 < \beta \leq \Delta = O(1)$, and K's vertex set S is contained within $[-\Delta, \Delta]^d$. It is an immediate consequence that $\text{width}_v(K) = \Theta(1)$ for all directions v, and hence it suffices to answer the membership query to an absolute error of $\Theta(\varepsilon)$.

Lemma 4(c) implies that we can answer ε-approximate width queries for K as the sum of two ε-approximate width queries to A and B. Therefore, our goal is to determine approximately if $O \in K$ using only approximate width queries to A and B. In order to do this, we look at the projective dual problem in which each point $p = (p_1, \dots, p_d) \in S$ is mapped to the hyperplane $p^* : x_d = p_1 x_1 + \dots + p_{d-1} x_{d-1} - p_d$. Let S^* denote the corresponding set of hyperplanes. The primal problem $O \in K$ is equivalent to the dual problem of determining whether the horizontal hyperplane $O^* : x_d = 0$ is sandwiched between the upper and lower envelopes of S^* (Figure 2(b)). Since the point c lies vertically above the origin and within K's interior, it follows that O^* cannot intersect the lower envelope. Therefore, it suffices to test whether O^* intersects the upper envelope.

The dual problem can be solved exactly by computing the minimum value y of the x_d-coordinate in the upper envelope and testing whether $y > 0$. In the primal, the value of y corresponds to the negated x_d-coordinate of the intersection of a facet F of the lower convex hull of K and a vertical line passing through the origin (see Figure 2). Let F's supporting hyperplane be denoted by $x_d = w_1 x_1 + \dots + w_{d-1} x_{d-1} - w_d$. Since K is sandwiched between two concentric balls of constant radii whose common center lies on this vertical line, it follows from simple geometry that this supporting hyperplane cannot be very steep. In particular, there exists $\alpha = O(1)$ such that $w_i \in [-\alpha, \alpha]$, for $i = 1, \dots, d-1$. In the dual, this means that the minimum value y is attained at a point whose first $d-1$ coordinates all lie within

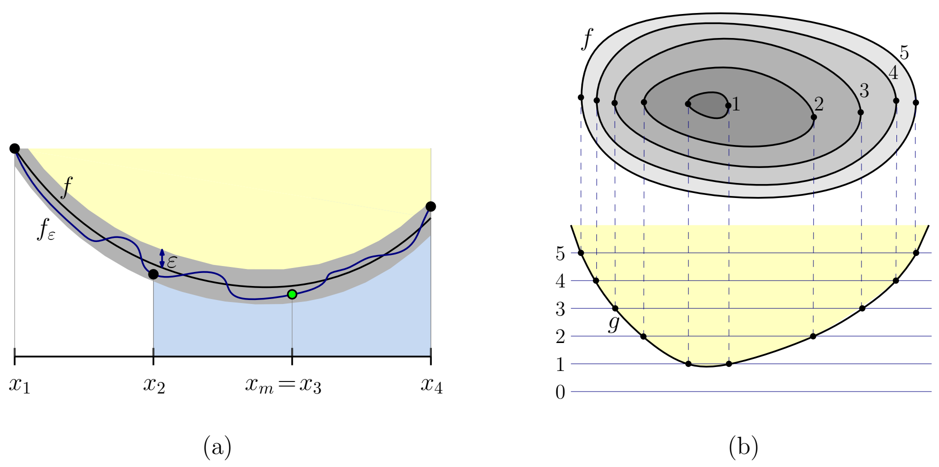


Figure 3 (a) One-dimensional convex minimization. (b) Higher-dimensional convex minimization.

$[-\alpha, \alpha]$. In approximating y, we will apply directional width queries only for directional vectors $v = (v_1, \ldots, v_d)$ whose first $d-1$ coordinates lie within $[-\alpha, \alpha]$ and $v_d = -1$. Thus, $\|v\| = O(1)$.

By Lemma 11, the duals of two points $p, q \in S$ returned by an exact directional width query $\text{width}_v(K)$ in the primal for a vector $v = (v_1, \ldots, v_{d-1}, -1)$ correspond to the two dual hyperplanes in the upper and lower envelopes of S^* that intersect the vertical line $x_i = v_i$ for $i = 1, \ldots, d-1$. Since queries are only applied to directions v where $\|v\| = O(1)$ and since $\text{width}_v(K) = \Theta(1)$ for all directions v, it follows from Lemma 11 that a relative error of ε in the directional width implies an absolute error of $O(\varepsilon)$ in the corresponding thickness. We can think of the upper envelope of S^* as defining the graph of a convex function over the domain $[-\alpha, \alpha]^{d-1}$. Since $S \subset [-\Delta, \Delta]^d$, the slopes of the hyperplanes in S^* are similarly bounded, and therefore this function has bounded slope. It follows that, for an appropriate $\varepsilon' = \Theta(\varepsilon)$, we can compute this function to an absolute error of ε at any $(v_1, \ldots, v_{d-1})$ by performing an (ε')-approximate directional width query on K for $v = (v_1, \ldots, v_{d-1}, -1)$. To complete the proof, it suffices to show that with $O(\text{polylog} \frac{1}{\varepsilon})$ such queries, it is possible to compute an absolute ε-approximation to y. We do this in the next section. ◀

3.1 Convex Minimization

The following lemma shows how to use binary search to solve a one-dimensional convex minimization problem approximately (see Figure 3(a)).

▶ **Lemma 14.** *Let $a, b \in \mathbb{R}$ and $\varepsilon \in \mathbb{R}+$ be real parameters. Let $f : [a, b] \to \mathbb{R}$ be a convex function with bounded slope and $f_\varepsilon : [a, b] \to \mathbb{R}$ be a function with $|f(x) - f_\varepsilon(x)| \leq \varepsilon$ for all $x \in [a, b]$. Let $x^* \in [a, b]$ be the value of x that minimizes $f(x)$. It is possible to determine a value x' with $f(x') - f(x^*) = O(\varepsilon)$ after $O(\log((b-a)/\varepsilon))$ evaluations of $f_\varepsilon(\cdot)$ and no evaluation of $f(\cdot)$.*

Proof. First, we present the recursive algorithm used to determine the value x'. If $b - a < \varepsilon$, then since the function has bounded slope, we simply return $x' = a$, as a valid answer.

Otherwise, we start by trisecting the interval $[a, b]$ and evaluate $f_\varepsilon(x)$ at the four endpoints x_1, x_2, x_3, x_4 of the subintervals (see Figure 3(a)). Let m denote the value i that minimizes $f_\varepsilon(x_i)$, breaking ties arbitrarily. To simplify the boundary cases, let $x_0 = a$ and $x_5 = b$.

We then invoke our algorithm recursively on the interval $[x_{m-1}, x_{m+1}]$ and store the value returned as x''. We return the value x among the two values x_m, x'' that minimizes $f_\varepsilon(x)$.

Since the length of the interval reduces by at least one third at each iteration, the number of recursive calls and therefore evaluations of $f_\varepsilon(\cdot)$ is $O(\log((b-a)/\varepsilon))$. Next, we show that $f(x') - f(x^*) = O(\varepsilon)$. By the convexity of f we have

$$f(x) \;\geq\; f(x_{m+1}) + 3(x - x_{m+1})(f(x_{m+1}) - f(x_m))/(b-a), \text{ for } x \geq x_{m+1}.$$

Using that $|f(x) - f_\varepsilon(x)| \leq \varepsilon$, we have

$$f(x) \;\geq\; f_\varepsilon(x_{m+1}) - \varepsilon + 3(x - x_{m+1})(f_\varepsilon(x_{m+1}) - f_\varepsilon(x_m) - 2\varepsilon)/(b-a), \text{ for } x \geq x_{m+1}.$$

Since $f_\varepsilon(x_m) \leq f_\varepsilon(x_{m+1})$, we have

$$f(x) \;\geq\; f_\varepsilon(x_m) - \varepsilon - 6\varepsilon(x - x_{m+1})/(b-a), \text{ for } x \geq x_{m+1}.$$

For x inside the interval $[a, b]$ we have $|x - x_{m+1}| \leq b - a$, and therefore

$$f(x) \;\geq\; f_\varepsilon(x_m) - 7\varepsilon, \text{ for } x_{m+1} \leq x \leq b.$$

The same argument is used to bound the case of $a \leq x \leq x_{m-1}$, obtaining

$$f(x) \;\geq\; f_\varepsilon(x_m) - 7\varepsilon, \text{ for } x \notin [x_{m-1}, x_{m+1}].$$

Either the minimum of $f(x)$ is inside the interval $[x_{m-1}, x_{m+1}]$ or not. If it is not, then the previous inequality shows that $f_\varepsilon(x_m)$ provides a good approximation, regardless of the value returned in the recursive call. If the minimum is inside the interval $[x_{m-1}, x_{m+1}]$, then the recursive call will provide a value result by an inductive argument. ◀

We are now ready to extend the result to arbitrary dimensions.

▶ **Lemma 15.** *Let $a, b \in \mathbb{R}$ and $\varepsilon \in \mathbb{R}+$ be real parameters. Let $f : [a, b]^d \to \mathbb{R}$ for a constant dimension d be a convex function with bounded slope and $f_\varepsilon : [a, b]^d \to \mathbb{R}$ be a function with $|f(x) - f_\varepsilon(x)| \leq \varepsilon$ for all $x \in [a, b]^d$. Let $x^* \in [a, b]^d$ be the value of x that minimizes $f(x)$. It is possible to determine a value x' with $f(x') - f(x^*) = O(\varepsilon)$ after $O(\log^d((b-a)/\varepsilon))$ evaluations of $f_\varepsilon(\cdot)$ and no evaluation of $f(\cdot)$.*

Proof. The minimum $f(x^*)$ can be written as

$$f(x^*) = \min_{x \in [a,b]^d} f(x) = \min_{x_1 \in [a,b]} \; \min_{\tilde{x} \in [a,b]^{d-1}} f(x_1, \tilde{x}).$$

Note that if $f(x)$ is a convex function with bounded slope, then so is the function $g : [a, b] \to \mathbb{R}$ (see Figure 3(b)) defined as

$$g(x_1) = \min_{\tilde{x} \in [a,b]^{d-1}} f(x_1, \tilde{x}).$$

The proof is based on induction on the dimension d. Since d is a constant, the number of induction steps is also a constant. The base case of $d = 1$ follows from Lemma 14. By the induction hypothesis, we can solve the $(d-1)$-dimensional instance to obtain a function $g'(x_1)$ such that

$$|g(x_1) - g'(x_1)| = O(\varepsilon).$$

Using Lemma 14 for the function $g'(\cdot)$, we obtain a value x' with $f(x') - f(x^*) = O(\varepsilon)$.

For the number of function evaluations $t(d)$ for a given dimension d we have

$t(1) = O(\log((b-a)/\varepsilon))$ and

$t(k) = t(1) \cdot t(k-1)$.

The recurrence easily solves to the desired

$t(d) = O(\log^d((b-a)/\varepsilon))$. ◀

By applying Lemma 15 to the dual problem defined in the proof of Lemma 13 (where f is the graph of the upper envelope of S^* and $[a,b] = [-\alpha, \alpha]$) with the augmented data structure from Lemma 5, we obtain Theorem 1 for the case when the input polytopes are represented by points. We will consider the case when the input polytopes are represented by halfspaces at the end of the next section.

4 Minkowski Sum Approximation

In this section, we will prove Theorems 2 and 3, as well as Theorem 1 for the case when the input polytopes are represented by halfspaces. Assume that we are given two polytopes A and B in the point representation, and we have computed the augmented approximate directional width data structures from Lemma 5 for each polytope. The objective is to obtain an ε-approximation of the Minkowski sum $A \oplus B$ of size $O(1/\varepsilon^{(d-1)/2})$ using these data structures. Our approach is to fatten $A \oplus B$ using Lemma 10 and then apply Dudley's construction [27] in order to obtain an approximation with $O(1/\varepsilon^{(d-1)/2})$ halfspaces. For completeness, we start by describing Dudley's algorithm.

Let $K \subset [-1,1]^d$ be a fat polytope of constant diameter. Dudley's algorithm obtains an ε-approximation represented by halfspaces as follows. Let D be a ball of radius $2\sqrt{d}$ centered at the origin. (Note that $K \subset D$.) Place a set W of $\Theta(1/\varepsilon^{(d-1)/2})$ points on the surface of D such that every point on the surface of D is within distance $O(\sqrt{\varepsilon})$ of some point in W. For each point $w \in W$, let w' be its nearest point on the boundary of K. We call these points *samples*. For each sample point w', take the supporting halfspace passing through w' that is orthogonal to the vector from w' to w. The approximation is defined as the intersection of these halfspaces (see Figure 4(a)).

Bronshteyn and Ivanov [16] presented a similar construction. Instead of approximating K by halfspaces, Bronshteyn and Ivanov's construction approximates K as the convex hull of the aforementioned set of samples[4] (see Figure 4(b)). In both constructions it is possible to tune the constant factors so that closest point queries need only be computed to within an absolute error of $\Theta(\varepsilon)$.

An approximate closest point query between a polytope K and a point p within constant distance from K can be reduced to computing an ε-approximation to the smallest radius ball centered at p that intersects K. This can be solved through binary search on the radius of this ball, where each probe involves determining whether K intersects a ball of some radius centered at p. Notice that the data structure for approximate polytope intersection from Section 3 only accesses the bodies through approximate directional width queries, besides the initial fattening transformation. By Lemma 4(c), given two preprocessed bodies A and

[4] Dudley's construction yields an outer approximation and Bronshteyn and Ivanov's yields inner approximation, but it is possible to convert both to the other type through standard techniques. For details, see Lemma 2.8 of the full version of [9].

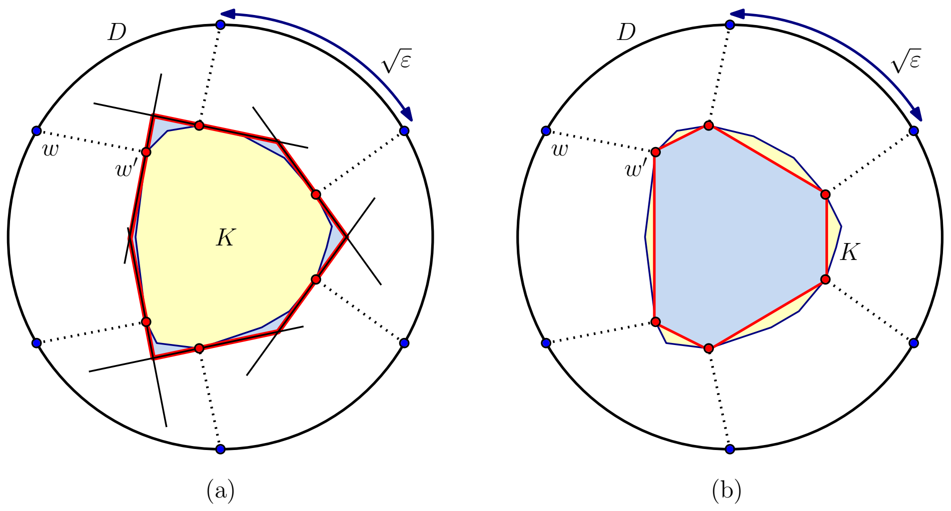


Figure 4 (a) Dudley's and (b) Bronshteyn and Ivanov's polytope approximations.

B, we can answer directional width queries on $A \oplus B$ through directional width queries on A and B individually. (In the case of a ball, no data structure is required.) Therefore, we can test intersection with a Minkowski sum $A \oplus B$, as long as we have augmented approximate directional width data structures for both A and B.

In order to establish Theorem 2 for the case when the input polytopes are represented by points, we apply the aforementioned binary search to simulate Dudley's construction. Each sample is obtained after $O(\log \frac{1}{\varepsilon})$ ε-approximate polytope intersection queries. The total running time is dominated by the preprocessing time of Lemma 5. Note that the output polytope may be represented by either points or halfspaces according to whether we use Dudley's or Bronshteyn and Ivanov's algorithm. To show that the input polytopes may be represented by halfspaces, we show how to efficiently convert between the two representations.

▶ **Lemma 16.** *Given an approximation parameter $\varepsilon > 0$ and a polytope $K \subset \mathbb{R}^d$ of size n (given either using a point or halfspace representation), we can obtain an ε-approximation of size $O(1/\varepsilon^{(d-1)/2})$ (in either representation, independent of the input representation) in $O(n \log \frac{1}{\varepsilon} + 1/\varepsilon^{(d-1)/2+\alpha})$ time, where $\alpha > 0$ is an arbitrarily small constant.*

Proof. The case when the input is represented by points is a trivial case of Theorem 2, where $B = \{O\}$. For the alternative case, it suffices to obtain an ε-approximation of the polar polytope after fattening. (For details see Lemma 2.9 of the full version of [9].) ◀

We remind the reader that Agarwal *et al.* [2] showed that the width of a convex body K is equal to the minimum distance from the origin to the boundary of the convex body $K \oplus (-K)$. To obtain Theorem 3, we compute Dudley's approximation of $K \oplus (-K)$ and then we determine the closest point to the origin among the $O(1/\varepsilon^{(d-1)/2})$ bounding hyperplanes of the approximation.

References

1 P. K. Agarwal, E. Flato, and D. Halperin. Polygon decomposition for efficient construction of Minkowski sums. *Comput. Geom. Theory Appl.*, 21(1):39–61, 2002.

2 P. K. Agarwal, L. J. Guibas, S. Har-Peled, A. Rabinovitch, and M. Sharir. Penetration depth of two convex polytopes in 3D. *Nordic J. of Computing*, 7(3):227–240, 2000.

3 P. K. Agarwal, S. Har-Peled, H. Kaplan, and M. Sharir. Union of random Minkowski sums and network vulnerability analysis. *Discrete Comput. Geom.*, 52(3):551–582, 2014.

4 P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. Assoc. Comput. Mach.*, 51:606–635, 2004.

5 P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Geometric approximation via coresets. In J. E. Goodman, J. Pach, and E. Welzl, editors, *Combinatorial and Computational Geometry*. MSRI Publications, 2005.

6 P. K. Agarwal, J. Matoušek, and S. Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Comput. Geom. Theory Appl.*, 1(4):189–201, 1992.

7 B. Aronov and M. Sharir. On translational motion planning of a convex polyhedron in 3-space. *SIAM J. Comput.*, 26(6):1785–1803, 1997.

8 S. Arya and T. M. Chan. Better ε-dependencies for offline approximate nearest neighbor search, Euclidean minimum spanning trees, and ε-kernels. In *Proc. 30th Annu. Sympos. Comput. Geom.*, pages 416–425, 2014.

9 S. Arya, G. D. da Fonseca, and D. M. Mount. Near-optimal ε-kernel construction and related problems. In *Proc. 33rd Internat. Sympos. Comput. Geom.*, pages 10:1–15, 2017. URL: `https://arxiv.org/abs/1604.01175`.

10 S. Arya, G. D. da Fonseca, and D. M. Mount. On the combinatorial complexity of approximating polytopes. *Discrete Comput. Geom.*, 58(4):849–870, 2017.

11 S. Arya, G. D. da Fonseca, and D. M. Mount. Optimal approximate polytope membership. In *Proc. 28th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 270–288, 2017.

12 S. Arya, G. D. da Fonseca, and D. M. Mount. Approximate polytope membership queries. *SIAM J. Comput.*, 47(1):1–51, 2018.

13 L. Barba and S. Langerman. Optimal detection of intersections between convex polyhedra. In *Proc. 26th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 1641–1654, 2015.

14 G. Barequet and S. Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *J. Algorithms*, 38(1):91–109, 2001.

15 J.-D. Boissonnat, E. De Lange, and M. Teillaud. Minkowski operations for satellite antenna layout. In *Proc. 13th Annu. Sympos. Comput. Geom.*, pages 67–76, 1997.

16 E. M. Bronshteyn and L. D. Ivanov. The approximation of convex sets by polyhedra. *Siberian Math. J.*, 16:852–853, 1976.

17 T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. *Internat. J. Comput. Geom. Appl.*, 12:67–85, 2002.

18 T. M. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Comput. Geom. Theory Appl.*, 35(1):20–35, 2006.

19 T. M. Chan. Applications of Chebyshev polynomials to low-dimensional computational geometry. In *Proc. 33rd Internat. Sympos. Comput. Geom.*, pages 26:1–15, 2017.

20 B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21(4):671–696, 1992.

21 B. Chazelle and D. P. Dobkin. Detection is easier than computation. In *Proc. 12th Annu. ACM Sympos. Theory Comput.*, pages 146–153, 1980.

22 B. Chazelle and D. P. Dobkin. Intersection of convex objects in two and three dimensions. *J. Assoc. Comput. Mach.*, 34:1–27, 1987.

23 B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *J. Algorithms*, 21:579–597, 1996.

24 M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2010.

25 D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theo. Comp. Sci.*, 27(3):241–253, 1983.

26 D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra—A unified approach. In *Proc. Internat. Colloq. Automata Lang. Prog.*, pages 400–413, 1990.

27 R. M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *J. Approx. Theory*, 10(3):227–236, 1974.

28 C. A. Duncan, M. T. Goodrich, and E. A. Ramos. Efficient approximation and optimization algorithms for computational metrology. In *Proc. Eighth Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 121–130, 1997.

29 E. Fogel, D. Halperin, and C. Weibel. On the exact maximum complexity of Minkowski sums of polytopes. *Discrete Comput. Geom.*, 42(4):654–669, 2009.

30 X. Guo, L. Xie, and Y. Gao. Optimal accurate Minkowski sum approximation of polyhedral models. *Advanced Intelligent Computing Theories and Applications. With Aspects of Theoretical and Methodological Issues*, pages 179–188, 2008.

31 D. Halperin, O. Salzman, and M. Sharir. Algorithmic motion planning. In J. E. Goodman, J. O'Rourke, and C. D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, Discrete Mathematics and its Applications. CRC Press, 2017.

32 S. Har-Peled, T. M. Chan, B. Aronov, D. Halperin, and J. Snoeyink. The complexity of a single face of a Minkowski sum. In *Proc. Seventh Canad. Conf. Comput. Geom.*, pages 91–96, 1995.

33 P. Jiménez, F. Thomas, and C. Torras. 3D collision detection: A survey. *Computers & Graphics*, 25(2):269–285, 2001.

34 F. John. Extremum problems with inequalities as subsidiary conditions. In *Studies and Essays Presented to R. Courant on his 60th Birthday*, pages 187–204. Interscience Publishers, Inc., New York, 1948.

35 A. Kaul and J. Rossignac. Solid-interpolating deformations: construction and animation of pips. *Computers & graphics*, 16(1):107–115, 1992.

36 M. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *Proc. of IMA conference on mathematics of surfaces*, volume 1, pages 602–608, 1998.

37 D. M. Mount. Geometric intersection. In J. E. Goodman, J. O'Rourke, and C. D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, Discrete Mathematics and its Applications. CRC Press, 2017.

38 D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. *Theo. Comp. Sci.*, 7(2):217–236, 1978.

39 J. O'Rourke. *Computational geometry in C.* Cambridge University Press, 1998.

40 L. Pachter and B. Sturmfels. *Algebraic statistics for computational biology*, volume 13. Cambridge University Press, 2005.

41 R. Schneider. *Convex bodies: The Brunn-Minkowski theory.* Cambridge University Press, 1993.

42 M. I. Shamos. Geometric complexity. In *Proc. Seventh Annu. ACM Sympos. Theory Comput.*, pages 224–233, 1975.

43 H. R. Tiwary. On the hardness of computing intersection, union and Minkowski sum of polytopes. *Discrete Comput. Geom.*, 40(3):469–479, 2008.

44 G. Varadhan and D. Manocha. Accurate Minkowski sum approximation of polyhedral models. *Graphical Models*, 68(4):343–355, 2006.

45 H. Yu, P. K. Agarwal, R. Poreddy, and K. R. Varadarajan. Practical methods for shape fitting and kinetic data structures using coresets. *Algorithmica*, 52(3):378–402, 2008.

On the Worst-Case Complexity of TimSort

Nicolas Auger
Université Paris-Est, LIGM (UMR 8049), UPEM, F77454 Marne-la-Vallée, France

Vincent Jugé
Université Paris-Est, LIGM (UMR 8049), UPEM, F77454 Marne-la-Vallée, France

Cyril Nicaud
Université Paris-Est, LIGM (UMR 8049), UPEM, F77454 Marne-la-Vallée, France

Carine Pivoteau
Université Paris-Est, LIGM (UMR 8049), UPEM, F77454 Marne-la-Vallée, France

Abstract

TimSort is an intriguing sorting algorithm designed in 2002 for Python, whose worst-case complexity was announced, but not proved until our recent preprint. In fact, there are two slightly different versions of TimSort that are currently implemented in Python and in Java respectively. We propose a pedagogical and insightful proof that the Python version runs in $\mathcal{O}(n \log n)$. The approach we use in the analysis also applies to the Java version, although not without very involved technical details. As a byproduct of our study, we uncover a bug in the Java implementation that can cause the sorting method to fail during the execution. We also give a proof that Python's TimSort running time is in $\mathcal{O}(n+n \log \rho)$, where ρ is the number of runs (i.e. maximal monotonic sequences), which is quite a natural parameter here and part of the explanation for the good behavior of TimSort on partially sorted inputs.

2012 ACM Subject Classification Theory of computation → Sorting and searching

Keywords and phrases Sorting algorithms, Merge sorting algorithms, TimSort, Analysis of algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.4

1 Introduction

TimSort is a sorting algorithm designed in 2002 by Tim Peters [9], for use in the Python programming language. It was thereafter implemented in other well-known programming languages such as Java. The algorithm includes many implementation optimizations, a few heuristics and some refined tuning, but its high-level principle is rather simple: The sequence S to be sorted is first decomposed greedily into monotonic runs (i.e. nonincreasing or nondecreasing subsequences of S as depicted on Figure 1), which are then merged pairwise according to some specific rules.

The idea of starting with a decomposition into runs is not new, and already appears in Knuth's NaturalMergeSort [6], where increasing runs are sorted using the same mechanism as in MergeSort. Other merging strategies combined with decomposition into runs appear in the literature, such as the MinimalSort of [10] (see also [2, 8] for other considerations on the same topic). All of them have nice properties: they run in $\mathcal{O}(n \log n)$ and even $\mathcal{O}(n + n \log \rho)$, where ρ is the number of runs, which is optimal in the model of sorting by comparisons [7], using the classical counting argument for lower bounds. And yet, among all these merge-based algorithms, TimSort was favored in several very popular programming languages, which suggests that it performs quite well in practice.

© Nicolas Auger, Vincent Jugé, Cyril Nicaud, and Carine Pivoteau;
licensed under Creative Commons License CC-BY
26th Annual European Symposium on Algorithms (ESA 2018).
Editors: Yossi Azar, Hannah Bast, and Grzegorz Herman; Article No. 4; pp. 4:1–4:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$$S = (\ \underbrace{12, 10, 7, 5}_{\text{first run}},\ \underbrace{7, 10, 14, 25, 36}_{\text{second run}},\ \underbrace{3, 5, 11, 14, 15, 21, 22}_{\text{third run}},\ \underbrace{20, 15, 10, 8, 5, 1}_{\text{fourth run}}\ )$$

Figure 1 A sequence and its *run decomposition* computed by TimSort: for each run, the first two elements determine if it is increasing or decreasing, then it continues with the maximum number of consecutive elements that preserves the monotonicity.

TimSort running time was implicitly assumed to be $\mathcal{O}(n \log n)$, but our unpublished preprint [1] contains, to our knowledge, the first proof of it. This was more than ten years after TimSort started being used instead of QuickSort in several major programming languages. The growing popularity of this algorithm invites for a careful theoretical investigation. In the present paper, we make a thorough analysis which provides a better understanding of the inherent qualities of the merging strategy of TimSort. Indeed, it reveals that, even without its refined heuristics,[1] this is an effective sorting algorithm, computing and merging runs on the fly, using only local properties to make its decisions.

As the analysis we made in [1] was a bit involved and clumsy, we first propose in Section 3 a new pedagogical and self-contained exposition that TimSort runs in $\mathcal{O}(n \log n)$ time, which we want both clear and insightful. Using the same approach, we also establish in Section 4 that it runs in $\mathcal{O}(n + n \log \rho)$, a question left open in our preprint and also in a recent work[2] on TimSort [4]. Of course, the first result follows from the second, but since we believe that each one is interesting on its own, we devote one section to each of them. Besides, the second result provides with an explanation to why TimSort is a very good sorting algorithm, worth considering in most situations where in-place sorting is not needed.

To introduce our last contribution, we need to look into the evolution of the algorithm: there are actually not one, but two main versions of TimSort. The first version of the algorithm contained a flaw, which was spotted in [5]: while the input was correctly sorted, the algorithm did not behave as announced (because of a broken invariant). This was discovered by De Gouw and his co-authors while trying to prove formally the correctness of TimSort. They proposed a simple way to patch the algorithm, which was quickly adopted in Python, leading to what we consider to be the real TimSort. This is the one we analyze in Sections 3 and 4. On the contrary, Java developers chose to stick with the first version of TimSort, and adjusted some tuning values (which depend on the broken invariant; this is explained in Sections 2 and 5) to prevent the bug exposed by [5]. Motivated by its use in Java, we explain in Section 5 how, at the expense of very complicated technical details, the elegant proofs of the Python version can be twisted to prove the same results for this older version. While working on this analysis, we discovered yet another error in the correction made in Java. Thus, we compute yet another patch, even if we strongly agree that the algorithm proposed and formally proved in [5] (the one currently implemented in Python) is a better option.

2 TimSort core algorithm

The idea of TimSort is to design a merge sort that can exploit the possible "non randomness" of the data, without having to detect it beforehand and without damaging the performances on random-looking data. This follows the ideas of adaptive sorting (see [7] for a survey on taking presortedness into account when designing and analyzing sorting algorithms).

[1] These heuristics are useful in practice, but do not change the worst-case complexity of the algorithm.
[2] In [4], the authors refined the analysis of [1] to obtain very precise bounds for the complexity of TimSort and of similar algorithms.

```
Algorithm 1: TimSort.                                              (Python 3.6.5)
  Input : A sequence $S$ to sort
  Result: The sequence $S$ is sorted into a single run, which remains on the stack.
  Note: The function merge_force_collapse repeatedly pops the last two runs on the
        stack $\mathcal{R}$, merges them and pushes the resulting run back on the stack.
1 runs ← a run decomposition of $S$
2 $\mathcal{R}$ ← an empty stack
3 while runs ≠ ∅ do                                       // main loop of TimSort
4     remove a run $r$ from runs and push $r$ onto $\mathcal{R}$
5     merge_collapse($\mathcal{R}$)
6 if height($\mathcal{R}$) ≠ 1 then            // the height of $\mathcal{R}$ is its number of runs
7     merge_force_collapse($\mathcal{R}$)
```

```
Algorithm 2: The merge_collapse procedure.                         (Python 3.6.5)
  Input : A stack of runs $\mathcal{R}$
  Result: The invariant of Equations (1) and (2) is established.
  Note: The runs on the stack are denoted by $\mathcal{R}[1] \ldots \mathcal{R}$[height($\mathcal{R}$)], from top to bottom.
        The length of run $\mathcal{R}$[i] is denoted by $\mathsf{r}_i$. The blue highlight indicates that the
        condition was not present in the original version of TimSort (this will be discussed
        in section 5).
1 while height($\mathcal{R}$) > 1 do
2     $n$ ← height($\mathcal{R}$) − 2
3     if ($n > 0$ and $\mathsf{r}_3 \leqslant \mathsf{r}_2 + \mathsf{r}_1$) or ($n > 1$ and $\mathsf{r}_4 \leqslant \mathsf{r}_3 + \mathsf{r}_2$) then
4         if $\mathsf{r}_3 < \mathsf{r}_1$ then
5             merge runs $\mathcal{R}[2]$ and $\mathcal{R}[3]$ on the stack
6         else merge runs $\mathcal{R}[1]$ and $\mathcal{R}[2]$ on the stack
7     else if $\mathsf{r}_2 \leqslant \mathsf{r}_1$ then
8         merge runs $\mathcal{R}[1]$ and $\mathcal{R}[2]$ on the stack
9     else break
```

The first feature of TimSort is to work on the natural decomposition of the input sequence into maximal runs. In order to get larger subsequences, TimSort allows both nondecreasing and decreasing runs, unlike most merge sort algorithms.

Then, the merging strategy of TimSort (Algorithm 1) is quite simple yet very efficient. The runs are considered in the order given by the run decomposition and successively pushed onto a stack. If some conditions on the lengths of the topmost runs of the stack are not satisfied after a new run has been pushed, this can trigger a series of merges between pairs of runs at the top or right under. And at the end, when all the runs in the initial decomposition have been pushed, the last operation is to merge the remaining runs two by two, starting at the top of the stack, to get a sorted sequence. The conditions on the stack and the merging rules are implemented in the subroutine called `merge_collapse` detailed in Algorithm 2. This is what we consider to be TimSort core mechanism and this is the main focus of our analysis.

Another strength of TimSort is the use of many effective heuristics to save time, such as ensuring that the initial runs are not to small thanks to an insertion sort or using a special technique called “galloping” to optimize the merges. However, this does not interfere with our analysis and we will not discuss this matter any further.

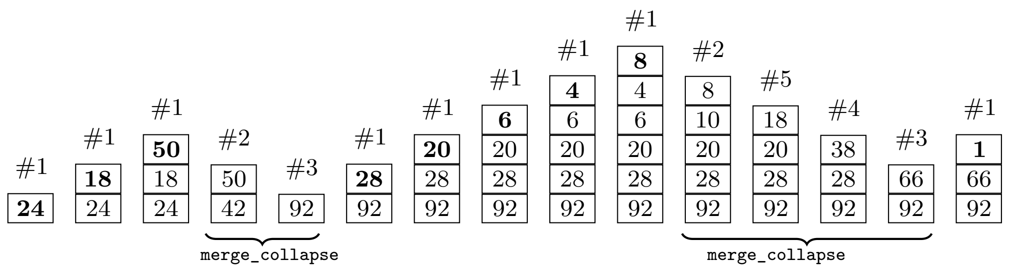


Figure 2 The successive states of the stack $\mathcal{R}$ (the values are the lengths of the runs) during an execution of the main loop of TimSort (Algorithm 1), with the lengths of the runs in **runs** being $(24, 18, 50, 28, 20, 6, 4, 8, 1)$. The label #1 indicates that a run has just been pushed onto the stack. The other labels refer to the different merges cases of `merge_collapse` as translated in Algorithm 3.

Let us have a closer look at Algorithm 2 which is a pseudo-code transcription of the `merge_collapse` procedure found in the latest version of Python (3.6.5). To illustrate its mechanism, an example of execution of the main loop of TimSort (lines 3-5 of Algorithm 1) is given in Figure 2. As stated in its note [9], Tim Peter's idea was that:

> "The thrust of these rules when they trigger merging is to balance the run lengths as closely as possible, while keeping a low bound on the number of runs we have to remember."

To achieve this, the merging conditions of `merge_collapse` are designed to ensure that the following invariant[3] is true at the end of the procedure:

$$\mathsf{r}_{i+2} > \mathsf{r}_{i+1} + \mathsf{r}_i, \quad (1)$$

$$\mathsf{r}_{i+1} > \mathsf{r}_i. \quad (2)$$

This means that the runs lengths r_i on the stack grow at least as fast as the Fibonacci numbers and, therefore, that the height of the stack stays logarithmic (see Lemma 6, section 3).

Note that the bound on the height of the stack is not enough to justify the $\mathcal{O}(n \log n)$ running time of TimSort. Indeed, without the smart strategy used to merge the runs "on the fly", it is easy to build an example using a stack containing at most two runs and that gives a $\Theta(n^2)$ complexity: just assume that all runs have length two, push them one by one onto a stack and perform a merge each time there are two runs in the stack.

We are now ready to proceed with the analysis of TimSort complexity. As mentioned earlier, Algorithm 2 does not correspond to the first implementation of TimSort in Python, nor to the current one in Java, but to the latest Python version. The original version will be discussed in details later, in Section 5.

3 TimSort runs in $\mathcal{O}(n \log n)$

At the first release of TimSort [9], a time complexity of $\mathcal{O}(n \log n)$ was announced with no element of proof given. It seemed to remain unproved until our recent preprint [1], where we provide a confirmation of this fact, using a proof which is not difficult but a bit tedious. This result was refined later in [4], where the authors provide lower and upper bounds, including explicit multiplicative constants, for different merge sort algorithms.

[3] Actually, in [9], the invariant is only stated for the 3 topmost runs of the stack.

Algorithm 3: TimSort: translation of Algorithm 1 and Algorithm 2.

```
Input : A sequence to $S$ to sort
Result: The sequence $S$ is sorted into a single run, which remains on the stack.
Note: At any time, we denote the height of the stack $\mathcal{R}$ by $h$ and its $i^{\text{th}}$ top-most run (for
      $1 \leqslant i \leqslant h$) by $R_i$. The length of this run is denoted by $r_i$.
1  runs ← the run decomposition of $S$
2  $\mathcal{R}$ ← an empty stack
3  while runs $\neq \emptyset$ do                                        // main loop of TimSort
4      remove a run $r$ from runs and push $r$ onto $\mathcal{R}$                        // #1
5      while true do
6          if $h \geqslant 3$ and $r_1 > r_3$ then merge the runs $R_2$ and $R_3$                // #2
7          else if $h \geqslant 2$ and $r_1 \geqslant r_2$ then merge the runs $R_1$ and $R_2$           // #3
8          else if $h \geqslant 3$ and $r_1 + r_2 \geqslant r_3$ then merge the runs $R_1$ and $R_2$      // #4
9          else if $h \geqslant 4$ and $r_2 + r_3 \geqslant r_4$ then merge the runs $R_1$ and $R_2$      // #5
10         else break
11 while $h \neq 1$ do merge the runs $R_1$ and $R_2$
```

Our main concern is to provide an insightful proof of the complexity of TimSort, in order to highlight how well designed is the strategy used to choose the order in which the merges are performed. The present section is more detailed than the following ones as we want it to be self-contained once TimSort has been translated into Algorithm 3 (see below).

As our analysis is about to demonstrate, in terms of worst-case complexity, the good performances of TimSort do not rely on the way merges are performed. Thus we choose to ignore their many optimizations and consider that merging two runs of lengths r and r' requires both $r + r'$ element moves and $r + r'$ element comparisons. Therefore, to quantify the running time of TimSort, we only take into account the number of comparisons performed.

▶ **Theorem 1.** *The running time of* TimSort *is* $\mathcal{O}(n \log n)$.

The first step consists in rewriting Algorithm 1 and Algorithm 2 in a form that is easier to deal with. This is done in Algorithm 3.

▶ **Claim 2.** *For any input, Algorithms 1 and 3 perform the same comparisons.*

Proof. The only difference is that Algorithm 2 was changed into the `while` loop of lines 5 to 10 in Algorithm 3. Observing the different cases, it is straightforward to verify that merges involving the same runs take place in the same order in both algorithms. Indeed, if $r_3 < r_1$, then $r_3 \leqslant r_1 + r_2$, and therefore line 5 is triggered in Algorithm 2, so that both algorithms merge the 2nd and 3rd runs. On the contrary, if $r_3 \geqslant r_1$, then both algorithms merge the 1st and 2nd runs if and only if $r_2 \leqslant r_1$ or $r_3 \leqslant r_1 + r_2$ (or $r_4 \leqslant r_2 + r_3$). ◀

▶ **Remark 3.** Proving Theorem 1 only requires analyzing the *main loop* of the algorithm (lines 3 to 10). Indeed, computing the run decomposition (line 1) can be done on the fly, by a greedy algorithm, in time linear in n, and the *final loop* (line 11) might be performed in the main loop by adding a fictitious run of length $n + 1$ at the end of the decomposition.

In the sequel, for the sake of readability, we also omit checking that h is large enough to trigger the cases #2 to #5. Once again, such omissions are benign, since adding fictitious runs of respective sizes $8n$, $4n$, $2n$ and n (in this order) at the beginning of the decomposition would ensure that $h \geqslant 4$ during the whole loop.

In Algorithm 3, we can see that the merges performed during Case #2 allow a very large run to be pushed and "absorbed" onto the stack without being merged all the way down, but by collapsing the stack under this run instead. Meanwhile, the purpose of Cases #3 to #5 is mainly to re-establish the invariant of Equations (1) and (2), ensuring an exponential growth of the run lengths within the stack (this duality is made even clearer in the proof of Section 4). Along this process, the cost of keeping the stack in good shape is compensated by the absorption of this large run, which naturally calls for an *amortized complexity* analysis.

To proceed with the core of our proof (that is the amortized analysis of the main loop), we now credit tokens to the elements of the input array, which are spent for comparisons. One token is paid for every comparison performed by the algorithm and each element is given $\mathcal{O}(\log n)$ tokens. Since the balance is always non-negative, we can conclude that at most $\mathcal{O}(n \log n)$ comparisons are performed, in total, during the main loop.

Elements of the input array are easily identified by their starting position in the array, so we consider them as well-defined and distinct entities (even if they have the same value). The *height* of an element is the number of runs that are below it in the stack: the elements belonging to the run R_i in the stack $(R_1, \ldots, R_h)$ have height $h - i$. To simplify the presentation, we also distinguish two kinds of tokens, the $\diamondsuit$-tokens and the $\heartsuit$-tokens, which can both be used to pay for comparisons.

Two $\diamondsuit$-tokens and one $\heartsuit$-token are credited to an element when it enters the stack (this is Case #1 of Algorithm 3) or when its height decreases: all the elements of R_1 are credited when R_1 and R_2 are merged, and all the elements of R_1 and R_2 are credited when R_2 and R_3 are merged. Tokens are spent to pay for comparisons, depending on the case triggered:

- Case #2: every element of R_1 and R_2 pays 1 $\diamondsuit$. This is enough to cover the cost of merging R_2 and R_3, since $r_2 + r_3 \leqslant r_2 + r_1$, as $r_3 < r_1$ in this case.
- Case #3: every element of R_1 pays 2 $\diamondsuit$. In this case $r_1 \geqslant r_2$ and the cost is $r_1 + r_2 \leqslant 2r_1$.
- Cases #4 and #5: every element of R_1 pays 1 $\diamondsuit$ and every element of R_2 pays 1 $\heartsuit$. The cost $r_1 + r_2$ is exactly the number of tokens spent.

▶ **Lemma 4.** *The balances of $\diamondsuit$-tokens and $\heartsuit$-tokens of each element remain non-negative throughout the main loop of* TimSort.

Proof. In all four cases #2 to #5, because the height of the elements of R_1 and possibly the height of those of R_2 decrease, the number of credited $\diamondsuit$-tokens after the merge is at least the number of $\diamondsuit$-tokens spent. The $\heartsuit$-tokens are spent in Cases #4 and #5 only: every element of R_2 pays one $\heartsuit$-token, and then belongs to the topmost run $\overline{R}_1$ of the new stack $\overline{\mathcal{S}} = (\overline{R}_1, \ldots, \overline{R}_{h-1})$ obtained after merging R_1 and R_2. Since $\overline{R}_i = R_{i+1}$ for $i \geqslant 2$, the condition of Case #4 implies that $\overline{r}_1 \geqslant \overline{r}_2$ and the condition of Case #5 implies that $\overline{r}_1 + \overline{r}_2 \geqslant \overline{r}_3$: in both cases, the next modification of the stack $\overline{\mathcal{S}}$ is another merge. This merge decreases the height of $\overline{R}_1$, and therefore decreases the height of the elements of R_2, who will regain one $\heartsuit$-token without losing any, since the topmost run of the stack never pays with $\heartsuit$-tokens. This proves that, whenever an element pay one $\heartsuit$-token, the next modification is another merge during which it regains its $\heartsuit$-token. This concludes the proof by direct induction. ◀

Let $h_{\max}$ be the maximum number of runs in the stack during the whole execution of the algorithm. Due to the crediting strategy, each element is given at most $2h_{\max}$ $\diamondsuit$-tokens and at most $h_{\max}$ $\heartsuit$-tokens in total. So we only need to prove that $h_{\max}$ is $\mathcal{O}(\log n)$ to complete the proof that the main loop running time is in $\mathcal{O}(n \log n)$. This fact is a consequence of TimSort's invariant established with a formal proof in the theorem prover KeY [3, 5]: at the end of any iteration of the main loop, we have $r_i + r_{i+1} < r_{i+2}$, for every $i \geqslant 1$ such that the run R_{i+2} exists.

For completeness, and because the formal proof is not meant to be read by humans, we sketch a "classical" proof of the invariant. It is not exactly the same statement as in [5], since our invariant holds at any time during the main loop: in particular we cannot say anything about R_1, which can have any length when a run has just been added. For technical reasons, and because it will be useful later on, we establish four invariants in our statement.

▶ **Lemma 5.** *At any step during the main loop of* TimSort*, we have* (i) $r_i + r_{i+1} < r_{i+2}$ *for* $i \in \{3, \ldots, h-2\}$*,* (ii) $r_2 < 3r_3$*,* (iii) $r_3 < r_4$ *and* (iv) $r_2 < r_3 + r_4$*.*

Proof. The proof is done by induction. It consists in verifying that, if all four invariants hold at some point, then they still hold when an update of the stack occurs in one of the five situations labeled #1 to #5 in the algorithm. This can be done by a straightforward case analysis. We denote by $\overline{\mathcal{S}} = (\overline{R}_1, \ldots, \overline{R}_{\overline{h}})$ the new state of the stack after the update:

- If Case #1 just occurred, a new run $\overline{R}_1$ was pushed. This implies that none of the conditions of Cases #2 to #5 hold in $\mathcal{S}$, otherwise merges would have continued. In particular, we have $r_1 < r_2 < r_3$ and $r_2 + r_3 < r_4$. As $\overline{r}_i = r_{i-1}$ for $i \geqslant 2$, and invariant (i) holds for $\mathcal{S}$, we have $\overline{r}_2 < \overline{r}_3 < \overline{r}_4$, and thus invariants (i) to (iv) hold for $\overline{\mathcal{S}}$.
- If one of the Cases #2 to #5 just occurred, we have $\overline{r}_2 = r_2 + r_3$ (in Case #2) or $\overline{r}_2 = r_3$ (in Cases #3 to #5). This implies that $\overline{r}_2 \leqslant r_2 + r_3$. As $\overline{r}_i = r_{i+1}$ for $i \geqslant 3$, and invariants (i) to (iv) hold for $\mathcal{S}$, we have $\overline{r}_2 \leqslant r_2 + r_3 < r_3 + r_4 + r_3 < 3r_4 = 3\overline{r}_3$, $\overline{r}_3 = r_4 \leqslant r_3 + r_4 < r_5 = \overline{r}_4$, and $\overline{r}_2 \leqslant r_2 + r_3 < r_3 + r_4 + r_3 < r_3 + r_5 < r_4 + r_5 = \overline{r}_3 + \overline{r}_4$. Thus, invariants (i) to (iv) hold for $\overline{\mathcal{S}}$. ◀

At this point, invariant (i) can be used to bound $h_{\max}$ from above.

▶ **Lemma 6.** *At any time during the main loop of* TimSort*, if the stack is* $(R_1, \ldots, R_h)$ *then we have* $r_2/3 < r_3 < r_4 < \ldots < r_h$ *and, for all* $i \geqslant j \geqslant 3$*, we have* $r_i > \sqrt{2}^{\,i-j-1} r_j$*. As a consequence, the number of runs in the stack is always* $\mathcal{O}(\log n)$*.*

Proof. By Lemma 5, we have $r_i + r_{i+1} < r_{i+2}$ for $3 \leqslant i \leqslant h-2$. Thus $r_{i+2} - r_{i+1} > r_i > 0$ and the sequence is increasing from index 4: $r_4 < r_5 < r_6 < \ldots < r_h$. The increasing sequence of the statement is then obtained using the invariants (ii) and (iii). Hence, for $j \geqslant 3$, we have $r_{j+2} > 2r_j$, from which one can get that $r_i > \sqrt{2}^{\,i-j-1} r_j$. In particular, if $h \geqslant 3$ then $r_h > \sqrt{2}^{\,h-4} r_3$, which yields that the number of runs is $\mathcal{O}(\log n)$ as $r_h \leqslant n$. ◀

Collecting all the above results is enough to prove Theorem 1. First, as mentioned in Remark 3, computing the run decomposition can be done in linear time. Then, we proved that the main loop requires $\mathcal{O}(nh_{\max})$ comparisons, by bounding from above the total number of tokens credited, and that $h_{\max} = \mathcal{O}(\log n)$, by showing that the run lengths grow at exponential speed. Finally, the final merges of line 11 might be taken care of by Remark 3, but they can also be dealt with directly:[4] if we start these merges with a stack $S = (R_1, \ldots, R_h)$, then every element of the run R_i takes part in $h + 1 - i$ merges at most, which proves that the overall cost of line 11 is $\mathcal{O}(n \log n)$. This concludes the proof of the theorem.

4 Refined analysis parametrized with the number of runs

A widely spread idea to explain why certain sorting algorithms perform better in practice than expected is that they are able to exploit presortedness [7]. This can be quantified in

[4] Relying on Remark 3 will be necessary only in the next section, where we need more precise computations.

$$\underbrace{\#1\ \#2\ \#2\ \#2}_{\text{starting seq.}}\ \underbrace{\#3\ \#2\ \#5\ \#2\ \#4\ \#2}_{\text{ending seq.}}\ \underbrace{\#1\ \#2\ \#2\ \#2\ \#2\ \#2}_{\text{starting seq.}}\ \underbrace{\#5\ \#2\ \#3\ \#3\ \#4\ \#2}_{\text{ending seq.}}$$

Figure 3 The decomposition of the encoding of an execution into starting and ending sequences.

many ways, the number of runs in the input sequence being one. Since this is the most natural parameter, we now consider the complexity of TimSort, according to it. We establish the following result, which was left open in [1, 4]:

▶ **Theorem 7.** *The complexity of* TimSort *on inputs of size n with ρ runs is $\mathcal{O}(n + n \log \rho)$.*

If $\rho = 1$, then no merge is to be performed, and the algorithm clearly runs in time linear in n. Hence, we assume below that $\rho \geqslant 2$, and we show that the complexity of TimSort is $\mathcal{O}(n \log \rho)$ in this case.

To obtain the $\mathcal{O}(n \log \rho)$ complexity, we need to distinguish several situations. First, consider the sequence of Cases #1 to #5 triggered during the execution of the main loop of TimSort. It can be seen as a word on the alphabet $\{\#1, \ldots, \#5\}$ that starts with #1, which completely encodes the execution of the algorithm. We split this word at every #1, so that each piece corresponds to an iteration of the main loop. Those pieces are in turn split into two parts, at the first occurrence of a symbol #3, #4 or #5. The first half is called a *starting sequence* and is made of a #1 followed by the maximal number of #2's. The second half is called an *ending sequence*, it starts with #3, #4 or #5 (or is empty) and it contains no occurrence of #1 (see Figure 3 for an example).

We treat starting and ending sequences separately in our analysis. The following lemma points out one of the main reasons TimSort is so efficient regarding the number of runs.

▶ **Lemma 8.** *The number of comparisons performed during all the starting sequences is $\mathcal{O}(n)$.*

Proof. More precisely, for a stack $\mathcal{S} = (R_1, \ldots, R_h)$, we prove that a starting sequence beginning with a push of a run R of length r onto $\mathcal{S}$ uses at most γr comparisons in total, where γ is the real constant $3\sqrt{2}\sum_{i \geqslant 0} i/\sqrt{2}^i$. After the push, the stack is $\overline{\mathcal{S}} = (R, R_1, \ldots, R_h)$ and, if the starting sequence contains $k \geqslant 1$ letters, i.e. $k-1$ occurrences of #2, then this sequence amounts to merging the runs R_1, R_2, …, R_k. Since no merge is performed if $k = 1$, we assume below that $k \geqslant 2$.

Looking closely at these runs, we compute that they require a total of

$$C = (k-1)r_1 + (k-1)r_2 + (k-2)r_3 + \ldots + r_k \leqslant \sum_{i=1}^{k} (k+1-i) r_i$$

comparisons. The last occurrence of Case #2 ensures that $r > r_k$, hence applying Lemma 6 to the stack $\overline{\mathcal{S}}$ shows that $r > \sqrt{2}^{k-i} r_i/3$ for all $i = 1, \ldots, k$. It follows that

$$C/r < 3 \sum_{i=2}^{k} (k+1-i)/\sqrt{2}^{k-i} < \gamma.$$

This concludes the proof, since each run is the beginning of exactly one starting sequence, and the sum of their lengths is n. ◀

We can now focus on the cost of ending sequences. Because the inner loop (line 5) of TimSort has already begun, during the corresponding starting sequence, we have some information on the length of the topmost run.

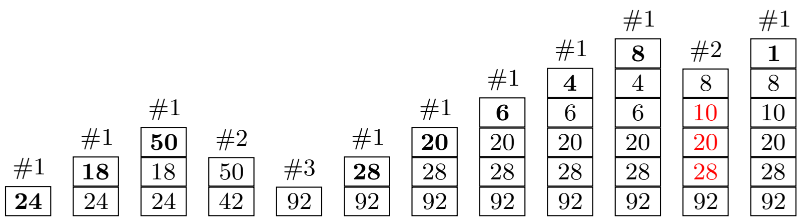


Figure 4 Execution of the main loop of Java's TimSort (Algorithm 3, without merge case #5, at line 9), with the lengths of the runs in `runs` being $(24, 18, 50, 28, 20, 6, 4, 8, 1)$. When the second to last run (of length 8) is pushed onto the stack, the while loop of line 5 stops after only one merge, breaking the invariant (in red), unlike what we see in Figure 2 using the Python version of TimSort.

▶ **Lemma 9.** *At any time during an ending sequence, including just before it starts and just after it ends, we have $r_1 < 3r_3$.*

Proof. The proof is done by induction. At the beginning of the ending sequence, the condition of #2 cannot be true, so $r_1 \leqslant r_3 < 3r_3$. Before any merge during an ending sequence, if the stack is $\mathcal{S} = (R_1, \ldots R_h)$, then we denote by $\overline{\mathcal{S}} = (\overline{R}_1, \ldots, \overline{R}_{h-1})$ the stack after that merge. If the invariant holds before the merge, and since $r_2 < r_3 + r_4$ and $r_3 < r_4$ by Lemma 5, we have $\overline{r}_1 = r_1 < 3r_3 < 3r_4 = 3\overline{r}_3$ in Case #2, and $\overline{r}_1 = r_1 + r_2 \leqslant r_3 + r_2 < r_3 + r_3 + r_4 < 3r_4 = 3\overline{r}_3$ in Cases #3 to #5 (since $r_1 \leqslant r_3$, as Case #2 does not apply), concluding the proof. ◀

In order to obtain a suitable upper bound for the merges that happen during ending sequences, we refine the analysis of the previous section. We still use $\diamondsuit$-tokens and $\heartsuit$-tokens to pay for comparisons when the stack is not too high, and we use different tokens otherwise:

- at the beginning of the algorithm, a common pool is credited with $24\,n$ $\clubsuit$-tokens,
- all elements are still credited two $\diamondsuit$-tokens and one $\heartsuit$-token when entering the stack,
- no token (of any kind) is credited nor spent during merges of starting sequences (the cost of such sequences is already taken care of by Lemma 9),
- if the stack has height less than $\kappa = \lceil 2 \log_2 \rho \rceil$, elements are credited $\diamondsuit$-tokens and $\heartsuit$-tokens and merges (of ending sequences) are paid in the same fashion as in Section 3,
- if the stack has height at least κ, then merges (of ending sequences) are paid using $\clubsuit$-tokens, and elements are not credited any token when a merge decreases their height.

By the analysis of the previous section, at most $\mathcal{O}(n\kappa)$ comparisons are paid with $\diamondsuit$-tokens and $\heartsuit$-tokens. Hence, using Remark 3, we complete the proof of Theorem 7 by checking that we initially credited enough $\clubsuit$-tokens. This is a direct consequence of the following lemma, since at most ρ merges are paid by $\clubsuit$-tokens.

▶ **Lemma 10.** *A merge performed during an ending sequence with a stack containing at least κ runs costs at most $24n/\rho$ comparisons.*

Proof. Lemmas 5 and 9 prove that $r_2 < 3r_3$ and $r_1 < 3r_3$. Since a merging step either merges R_1 and R_2, or R_2 and R_3, it requires at most $6r_3$ comparisons. By Lemma 6, we have $r_h \geqslant \sqrt{2}^{h-4} r_3$, whence $6r_3 \leqslant 24\sqrt{2}^{-h} r_h \leqslant 24n\sqrt{2}^{-\kappa} \leqslant 24n/\rho$. ◀

5 About the Java version of TimSort

Algorithm 2 (and therefore Algorithm 3) does not correspond to the original TimSort. Before release 3.4.4 of Python, the second part of the condition (in blue) in the test at line 3 of `merge_collapse` (and therefore merge case #5 of Algorithm 3) was missing. This version

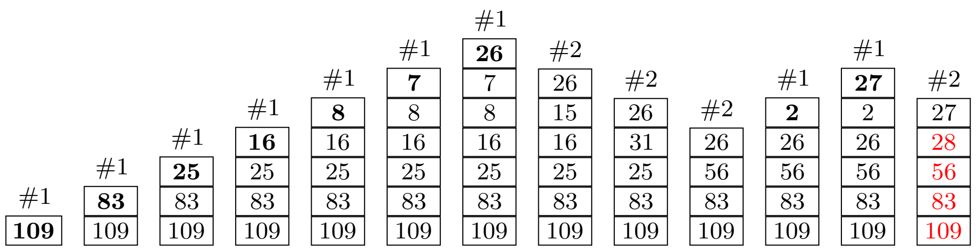


Figure 5 Execution of the main loop of the Java version of TimSort (without merge case #5, at line 9 of Algorithm 3), with the lengths of the runs in `runs` being $(109, 83, 25, 16, 8, 7, 26, 2, 27)$. When the algorithm stops, the invariant is violated twice, for consecutive runs (in red).

of the algorithm worked fine, meaning that it did actually sort arrays, but the invariant given by Equation (1) did not hold. Figure 4 illustrates the difference caused by the missing condition when running Algorithm 3 on the same input as in Figure 2.

This was discovered by de Gouw *et al.* [5] when trying to prove the correctness of the Java implementation of TimSort (which is the same as in the earlier versions of Python). And since the Java version of the algorithm uses the (wrong) invariant to compute the maximum size of the stack used to store the runs, the authors were able to build a sequence of runs that causes the Java implementation of TimSort to crash. They proposed two solutions to fix TimSort: reestablish the invariant, which led to the current Python version, or keep the original algorithm and compute correct bounds for the stack size, which is the solution that was chosen in Java 9 (note that this is the second time these values had to be changed). To do the latter, the developers used the claim in [5] that the invariant cannot be violated for two consecutive runs on the stack, which turns out to be false,[5] as illustrated in Figure 5. Thus, it is still possible to cause the Java implementation to fail: it uses a stack of runs of size at most 49 and we were able to compute an example requiring a stack of size 50 (see `http://igm.univ-mlv.fr/~pivoteau/Timsort/Test.java`), causing an error at runtime in Java's sorting method.

Even if the bug we highlighted in Java's TimSort is very unlikely to happen, this should be corrected. And, as advocated by de Gouw *et al.* and Tim Peters himself,[6] we strongly believe that the best solution would be to correct the algorithm as in the current version of Python, in order to keep it clean and simple. However, since this is the implementation of Java's sort for the moment, there are two questions we would like to tackle: Does the complexity analysis holds without the missing condition? And, can we compute an actual bound for the stack size? We first address the complexity question. It turns out that the missing invariant was a key ingredient for having a simple and elegant proof.

▶ **Proposition 11.** *At any time during the main loop of Java's* TimSort*, if the stack of runs is* $(R_1, \ldots, R_h)$ *then we have* $r_3 < r_4 < \ldots < r_h$ *and, for all* $i \geqslant 3$*, we have* $(2 + \sqrt{7})r_i \geqslant r_2 + \ldots + r_{i-1}$.

Proof ideas. The proof of Proposition 11 is much more technical and difficult than insightful, and therefore we just summarize its main steps. As in previous sections, this proof relies on several inductive arguments, using both inductions on the number of merges performed,

[5] This is the consequence of a small error in the proof of their Lemma 1. The constraint $C_1 > C_2$ has no reason to be. Indeed, in our example, we have $C_1 = 25$ and $C_2 = 31$.

[6] Here is the discussion about the correction in Python: `https://bugs.python.org/issue23515`.

on the stack size and on the run lengths. The inequalities $r_3 < r_4 < \ldots < r_h$ come at once, hence we focus on the second part of Proposition 11.

Since separating starting and ending sequences was useful in Section 4, we first introduce the notion of *stable* stacks: a stack $\mathcal{S}$ is stable if, when operating on the stack $\mathcal{S} = (R_1, \ldots, R_h)$, Case #1 is triggered (i.e. Java's TimSort is about to perform a *run push* operation).

We also call *obstruction indices* the integers $i \geqslant 3$ such that $r_i \leqslant r_{i-1} + r_{i-2}$: although they do not exist in Python's TimSort, they may exist, and even be consecutive, in Java's TimSort. We prove that, if $i-k, i-k+1, \ldots, i$ are obstruction indices, then the stack sizes $r_{i-k-2}, \ldots, r_i$ grow "at linear speed". For instance, in the last stack of Figure 5, obstruction indices are 4 and 5, and we have $r_2 = 28$, $r_3 = r_2 + 28$, $r_4 = r_3 + 27$ and $r_5 = r_4 + 26$.

Finally, we study so-called *expansion functions*, i.e. functions $f : [0,1] \mapsto \mathbb{R}$ such that, for every stable stack $\mathcal{S} = (R_1, \ldots, R_h)$, we have $r_2 + \ldots + r_{h-1} \leqslant r_h f(r_{h-1}/r_h)$. We exhibit an explicit function f such that $f(x) \leqslant 2 + \sqrt{7}$ for all $x \in [0,1]$, and we prove by induction on r_h that f is an expansion function, from which we deduce Proposition 11. ◀

Once Proposition 11 is proved, we easily recover the following variant of Lemmas 6 and 9.

▶ **Lemma 12.** *At any time during the main loop of Java's* TimSort*, if the stack is* $(R_1, \ldots, R_h)$ *then we have* $r_2/(2+\sqrt{7}) \leqslant r_3 < r_4 < \ldots < r_h$ *and, for all* $i \geqslant j \geqslant 3$, *we have* $r_i \geqslant \delta^{i-j-4} r_j$, *where* $\delta = \big(5/(2+\sqrt{7})\big)^{1/5} > 1$. *Furthermore, at any time during an ending sequence, including just before it starts and just after it ends, we have* $r_1 \leqslant (2+\sqrt{7}) r_3$.

Proof. The inequalities $r_2/(2+\sqrt{7}) \leqslant r_3 < r_4 < \ldots < r_h$ are just a (weaker) restatement of Proposition 11. Then, for $j \geqslant 3$, we have $(2+\sqrt{7}) r_{j+5} \geqslant r_j + \ldots + r_{j+4} \geqslant 5 r_j$, i.e. $r_{j+5} \geqslant \delta^5 r_j$, from which one gets that $r_i \geqslant \delta^{i-j-4} r_j$.

Finally, we prove by induction that $r_1 \leqslant (2+\sqrt{7}) r_3$ during ending sequences. First, when the ending sequence starts, $r_1 < r_3 \leqslant (2+\sqrt{7}) r_3$. Before any merge during this sequence, if the stack is $\mathcal{S} = (R_1, \ldots R_h)$, then we denote by $\overline{\mathcal{S}} = (\overline{R}_1, \ldots, \overline{R}_{h-1})$ the stack after the merge. If the invariant holds before the merge, in Case #2, we have $\overline{r}_1 = r_1 \leqslant (2+\sqrt{7}) r_3 \leqslant (2+\sqrt{7}) r_4 = (2+\sqrt{7}) \overline{r}_3$; and using Proposition 11 in Cases #3 and #4, we have $\overline{r}_1 = r_1 + r_2$ and $r_1 \leqslant r_3$, hence $\overline{r}_1 = r_1 + r_2 \leqslant r_2 + r_3 \leqslant (2+\sqrt{7}) r_4 = (2+\sqrt{7}) \overline{r}_3$, concluding the proof. ◀

We can then recover a proof of complexity for the Java version of TimSort, by following the same proof as in Sections 3 and 4, but using Lemma 12 instead of Lemmas 6 and 9.

▶ **Theorem 13.** *The complexity of Java's* TimSort *on inputs of size* n *with* ρ *runs is* $\mathcal{O}(n + n \log \rho)$.

Another question is that of the stack size requirements of Java's TimSort, i.e. computing $h_{\max}$. A first result is the following immediate corollary of Lemma 12.

▶ **Corollary 14.** *On an input of size* n, *Java's* TimSort *will create a stack of runs of maximal size* $h_{\max} \leqslant 7 + \log_\delta(n)$, *where* $\delta = \big(5/(2+\sqrt{7})\big)^{1/5}$.

Proof. At any time during the main loop of Java's TimSort on an input of size n, if the stack is $(R_1, \ldots, R_h)$ and $h \geqslant 3$, it follows from Lemma 12 that $n \geqslant r_h \geqslant \delta^{h-7} r_3 \geqslant \delta^{h-7}$. ◀

Unfortunately, for integers smaller than 2^{31}, Corollary 14 only proves that the stack size will never exceed 347. However, in the comments of Java's implementation of TimSort,[7]

[7] Comment at line 168: `http://igm.univ-mlv.fr/~pivoteau/Timsort/TimSort.java`.

there is a remark that keeping a short stack is of some importance, for practical reasons, and that the value chosen in Python– 85 – is "too expensive". Thus, in the following, we go to the extent of computing the optimal bound. It turns out that this bound cannot exceed 86 for such integers. This bound could possibly be refined slightly, but definitely not to the point of competing with the bound that would be obtained if the invariant of Equation (1) were correct. Once more, this suggests that implementing the new version of TimSort in Java would be a good idea, as the maximum stack height is smaller in this case.

▶ **Theorem 15.** *On an input of size n, Java's* TimSort *will create a stack of runs of maximal size $h_{\max} \leqslant 3 + \log_\Delta(n)$, where $\Delta = (1+\sqrt{7})^{1/5}$. Furthermore, if we replace Δ by any real number $\Delta' > \Delta$, the inequality fails for all large enough n.*

Proof ideas. The first part of Theorem 15 is proved as follows. Ideally, we would like to show that $r_{i+j} \geqslant \Delta^j r_i$ for all $i \geqslant 3$ and some fixed integer j. However, these inequalities do not hold for all i. Yet, we prove that they hold if $i+2$ and $i+j+2$ are not obstruction indices and if $i+j+1$ is an obstruction index. It follows quickly that $r_h \geqslant \Delta^{h-3}$.

The optimality of Δ is much more difficult to prove. It turns out that the constants $2+\sqrt{7}$, $(1+\sqrt{7})^{1/5}$, and the expansion function referred to in the proof of Proposition 11 were constructed as least fixed points of non-decreasing operators, although this construction needed not be explicit for using these constants and function. Hence, we prove that Δ is optimal by inductively constructing sequences of run lengths that show that $\limsup\{\log(r_h)/h\} \geqslant \Delta$; much care is required for proving that our constructions are indeed feasible. ◀

6 Conclusion

At first, when we learned that Java's QuickSort had been replaced by a variant of MergeSort, we thought that this new algorithm – TimSort – should be really fast and efficient in practice, and that we should look into its average complexity to confirm this from a theoretical point of view. Then, we realized that its worst-case complexity had not been formally established yet and we first focused on giving a proof that it runs in $\mathcal{O}(n \log n)$, which we wrote in a preprint [1]. In the present article, we simplify this preliminary work and provide a short, simple and self-contained proof of TimSort's complexity, which sheds some light on the behavior of the algorithm. Based on this description, we were also able to answer positively a natural question, which was left open so far: does TimSort runs in $\mathcal{O}(n + n \log \rho)$, where ρ is the number of runs? We hope our theoretical work highlights that TimSort is actually a very good sorting algorithm. Even if all its fine-tuned heuristics are removed, the dynamics of its merges, induced by a small number of local rules, results in a very efficient global behavior, particularly well suited for *almost sorted* inputs.

Besides, we want to stress the need for a thorough algorithm analysis, in order to prevent errors and misunderstandings. As obvious as it may sound, the three consecutive mistakes on the stack height in Java illustrate perfectly how the best ideas can be spoiled by the lack of a proper complexity analysis.

Finally, following [5], we would like to emphasize that there seems to be no reason not to use the recent version of TimSort, which is efficient in practice, formally certified and whose optimal complexity is easy to understand.

References

1 Nicolas Auger, Cyril Nicaud, and Carine Pivoteau. Merge strategies: From Merge Sort to TimSort. Research Report hal-01212839, hal, 2015. URL: `https://hal-upec-upem.archives-ouvertes.fr/hal-01212839`.

2 Jérémy Barbay and Gonzalo Navarro. On compressing permutations and adaptive sorting. *Theor. Comput. Sci.*, 513:109–123, 2013. `doi:10.1016/j.tcs.2013.10.019`.

3 Bernhard Beckert, Reiner Hähnle, and Peter H Schmitt. *Verification of object-oriented software: The KeY approach.* Springer-Verlag, 2007.

4 Sam Buss and Alexander Knop. Strategies for stable merge sorting. Research Report abs/1801.04641, arXiv, 2018. URL: `http://arxiv.org/abs/1801.04641`.

5 Stijn De Gouw, Jurriaan Rot, Frank S de Boer, Richard Bubel, and Reiner Hähnle. OpenJDK's Java.utils.Collection.sort() is broken: The good, the bad and the worst case. In *International Conference on Computer Aided Verification*, pages 273–289. Springer, 2015.

6 Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching.* Addison Wesley Longman Publish. Co., Redwood City, CA, USA, 1998.

7 Heikki Mannila. Measures of presortedness and optimal sorting algorithms. *IEEE Trans. Computers*, 34(4):318–325, 1985. `doi:10.1109/TC.1985.5009382`.

8 J. Ian Munro and Sebastian Wild. Nearly-optimal mergesorts: Fast, practical sorting methods that optimally adapt to existing runs. In Hannah Bast Yossi Azar and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 63:1–63:15, 2018.

9 Tim Peters. Timsort description, accessed june 2015. URL: `http://svn.python.org/projects/python/trunk/Objects/listsort.txt`.

10 Tadao Takaoka. Partial solution and entropy. In Rastislav Královič and Damian Niwiński, editors, *Mathematical Foundations of Computer Science 2009*, pages 700–711, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

A New and Improved Algorithm for Online Bin Packing

János Balogh
Department of Applied Informatics, Gyula Juhász Faculty of Education,
University of Szeged, Hungary
balogh@jgypk.u-szeged.hu

József Békési
Department of Applied Informatics, Gyula Juhász Faculty of Education,
University of Szeged, Hungary
bekesi@jgypk.u-szeged.hu

György Dósa
Department of Mathematics, University of Pannonia, Veszprém, Hungary
dosagy@almos.vein.hu

Leah Epstein
Department of Mathematics, University of Haifa, Haifa, Israel
lea@math.haifa.ac.il

Asaf Levin
Faculty of Industrial Engineering and Management, The Technion, Haifa, Israel
levinas@ie.technion.ac.il

Abstract

We revisit the classic online bin packing problem studied in the half-century. In this problem, items of positive sizes no larger than 1 are presented one by one to be packed into subsets called *bins* of total sizes no larger than 1, such that every item is assigned to a bin before the next item is presented. We use online partitioning of items into classes based on sizes, as in previous work, but we also apply a new method where items of one class can be packed into more than two types of bins, where a bin type is defined according to the number of such items grouped together. Additionally, we allow the smallest class of items to be packed in multiple kinds of bins, and not only into their own bins. We combine this with the approach of packing of sufficiently big items according to their exact sizes. Finally, we simplify the analysis of such algorithms, allowing the analysis to be based on the most standard weight functions. This simplified analysis allows us to study the algorithm which we defined based on all these ideas. This leads us to the design and analysis of the first algorithm of asymptotic competitive ratio strictly below 1.58, specifically, we break this barrier by providing an algorithm AH (Advanced Harmonic) whose asymptotic competitive ratio does not exceed 1.57829.

Our main contribution is the introduction of the simple analysis based on weight function to analyze the state of the art online algorithms for the classic online bin packing problem. The previously used analytic tool named *weight system* was too complicated for the community in this area to adjust it for other problems and other algorithmic tools that are needed in order to improve the current best algorithms. We show that the weight system based analysis is not needed for the analysis of the current algorithms for the classic online bin packing problem. The importance of a simple analysis is demonstrated by analyzing several new features together with all existing techniques, and by proving a better competitive ratio than the previously best one.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Bin packing, online algorithms, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.5

© Janos Balogh, Jozsef Bekesi, Gyorgy Dosa, Leah Epstein, and Asaf Levin;
licensed under Creative Commons License CC-BY

1 Introduction

Bin packing [5, 6] is the problem of packing a set of items of rational sizes in $(0, 1]$ into subsets of items, which are called bins, of total sizes no larger than 1. In the offline variant the list of items is given as a set, and in the online environment items are presented one by one and each item has to be packed into a bin irrevocably before the next item is presented.

For an algorithm A, we denote its cost, that is, the number of used bins in its packing on an input I by $A(I)$. The cost of an optimal solution OPT, for the same input, is denoted by $OPT(I)$. The *asymptotic approximation ratio* allows to compare the costs for inputs for which the optimal cost is sufficiently large. The asymptotic approximation ratio of A is defined as follows. $R_A = \lim_{N\to\infty} \left(\sup_{I:OPT(I)\geq N} \frac{A(I)}{OPT(I)} \right)$. In this paper we only consider the asymptotic approximation ratio, which is the common measure for bin packing algorithms. Thus we use the term approximation ratio throughout the paper, with the meaning of asymptotic approximation ratio. Moreover, the term *competitive ratio* often replaces the term "approximation ratio" in cases where online algorithms are considered. We will use this term for the asymptotic measure. When we discuss the absolute measure $\sup_I \frac{A(I)}{OPT(I)}$ (the absolute approximation ratio or the absolute competitive ratio), we will mention this explicitly. A standard method for proving an upper bound for the asymptotic approximation ratio or the asymptotic competitive ratio for an algorithm A is to show the existence of a constant $C \geq 0$ independent of the input, such that for any input I, $A(I) \leq R \cdot OPT(I) + C$ (and then the value of the asymptotic measure is at most R). Most work on upper bounds on the asymptotic competitive ratio provide in fact an upper bound using this last method, and we will follow this approach as well.

For the offline problem, algorithms with an approximation ratio of $1 + \varepsilon$ can be designed [10, 17, 9, 13] for any $\varepsilon > 0$. If the first definition is used, a 1-approximation is known [17], where the cost of the solution computed by the algorithm is $OPT(I) + o(OPT(I))$ (see also recent work on improving the sub-linear function of $OPT(I)$ [21, 12]).

The classic bin packing problem, which we study here, was presented in the early 1970's [25, 14, 15, 16]. It was introduced as an offline problem, but many of the algorithms initially proposed for it were in fact online. Johnson [14, 15] defined and analyzed the simple algorithm Next Fit (NF), which tries to pack the next item into the last bin that was used for packing, if such a bin exists (in which case such a bin is called "active") and the item can be packed there, and otherwise it opens a new bin for the item. The competitive ratio of this algorithm is 2 [14, 15]. Any Fit (AF) algorithms, as opposed to the behavior of NF which only tests at most one active bin for feasibility of packing a new item there, pack a new item into a nonempty bin unless this is impossible (in which case a new bin is opened). Such algorithms have competitive ratios of at most 2. Next, consider a sub-class of algorithms where one may not select a bin with smallest total size of currently packed items for packing a new item, unless this minimum is not unique or this is the only bin that can accommodate the new item except for an empty bin. The last class of algorithms is called Almost Any Fit (AAF), and they have competitive ratios of 1.7 [16, 15]. A well-known algorithm, which is in fact a special case of AAF is Best Fit (BF), which always chooses the fullest bin where the new item can be packed. First Fit (FF) is another important special case of AF (but not of AAF) which selects a minimum index bin for each new item (where it can be packed). The competitive ratio of FF is 1.7 [16, 7].

The pre-sorted versions of these algorithms, called NFD, FFD, BFD, and AFD, were studied as well. Here items are still presented one by one, but they are sorted in a non increasing order of sizes. For example, the approximation ratio of NFD is (approximately) 1.69103 [2] and that of FFD is $\frac{11}{9} \approx 1.22222$ [14]. For AFD in general, the approximation ratio is at most 1.25 [14, 15, 16]. These pre-sorted variants are not online algorithms.

We design and analyze a new algorithm AH (Advanced Harmonic) for online bin packing, and show that its competitive ratio does not exceed 1.57828956. This is the first algorithm whose asymptotic competitive ratio is below 1.58. We use a new type of analysis of algorithms which allows us to split the analysis into cases, while for every case we define only three values (and even just one value in a large number of cases), and based on those we calculate weights for items. The analysis is split into cases in recent previous work as well, but the analysis of each case is much more difficult. Items are partitioned into classes according to sizes. As in previous work, we sometimes do not pack the maximum number of items of some class into a bin, and leave space for items of another class (possibly arriving later). One new feature of AH is that in previous papers, in the algorithms there were at most two options for every class. For any given class, one option was a bin with the maximum number of items of this class fitting into a bin. For some of the classes there was a second option consisted of a very small number of items from this class (with reserved spaces for items of another class, possibly arriving later). We allow intermediate values as well with more than two options for some classes and not only two kinds of bins for a given class.

We use simple weight functions for the analysis, rather than the much more complicated tool called *weight systems* [23]. Weight functions are an auxiliary tool used for the analysis of bin packing (and other) algorithms (this technique is also called dual fitting). In this method, a weight is defined for each item (usually, based on its size, and sometimes it is also based on its role in the packing). If there are multiple kinds of outputs, it is possible to define a weight function for each one of them. The total weight of items is then used to compare the numbers of bins in the output of the algorithm and in an optimal solution. The list of weights of one item for different output types, also called scenarios, can be seen as a vector associated with the item. Thus, the weights can be seen as one function from the items to vectors whose dimension is the number of scenarios. Briefly, a weight system is a generalization where the weight function also maps items (or item sizes) to vectors, but in order to compute the weight of some item for a given scenario, another function, called a consolidation function, is used. This last function is a piecewise linear function (mapping real vectors to reals). The slightly simplified approach is to use convex combinations of weights according to subsets of scenarios. It has not been proved that weight systems are a stronger tool than just weights defined for the different scenarios. However, for simple weights every scenario can be analyzed independently from other scenarios. We exploit the simplicity of weight functions to obtain a clean and full analysis, which is easier to implement and verify (compared to the analysis resulting from weight systems). The main advantage is that every case is analyzed in a separate calculation using a standard knapsack solver without considering any other cases at that time. This simplicity allows us to analyze the new features that we introduce. Obviously, as these are cases for one algorithm, they have a common set of parameters, but once the algorithm has been fixed, there is no connection between the various cases.

The significance of our approach is that we combine many existing methods, including that of Babel et al. [1] (recently used by Heydrich and van Stee [22, 11] for classic bin packing), adding several new features, and applying a simple analysis, which can be verified easily and is robust to further changes of the algorithm. We define the action of our algorithm AH, we prove a number of invariants and properties of AH in detail, and then we provide the specific parameters and compact representations of the lists of weights. For every possible output type and scenario, there is a small number of values used for the calculation of weights for it that we choose based on solving an auxiliary linear program. We also provide explicit lists of weights calculated based on the values and the parameters.

To explain the new features of our work, we discuss the harmonic type algorithms. Already in much of the previous work on online algorithms for bin packing, items were partitioned into classes by size. The simplest such classification is based on harmonic numbers, leading to the Harmonic algorithm of Lee and Lee [18]. In the harmonic algorithm of index k (for an integer parameter $k \geq 2$), subset j is the intersection of the input and $(\frac{1}{j+1}, \frac{1}{j}]$ (where $1 \leq j \leq k-1$), and subset k of tiny items is the intersection of the input and $(0, \frac{1}{k}]$.

In these algorithms each subset is packed independently from other subsets using NF (so for $j \leq k-1$, any bin for subset j, except for possibly the last such bin, has j items, but for subset k, every bin except for the last bin for this subset has a total size of items above $\frac{k-1}{k}$), and for k growing to infinity, the resulting competitive ratio is approximately 1.69103 [18]. The drawback of those algorithms is that bins of subsets with small values of j can be packed with small sizes of items (for example, a bin of subset 2 may have total size just above $\frac{2}{3}$ and a bin of subset 1 may have just one item of size just above $\frac{1}{2}$).

The first idea which comes to mind is to try to combine items of those two subset into common bins. However, if items of class 2 arrive first, one cannot just pack them one per bin, as this immediately leads to a competitive ratio of 2 (if no items of subset 1 arrive afterwards). Lee and Lee [18] proposed the following method to overcome this. A fixed fraction of items of subset 2 (up to rounding errors) is packed one per bin and the remaining items are packed in pairs. Thus, there are two kinds of bins for subset 2. The items we refer to here can only be sufficiently small items, so there is a threshold $\Delta \in (\frac{1}{2}, \frac{2}{3})$ such that items of sizes in $(\Delta, 1]$ and $(1-\Delta, \frac{1}{2}]$ are packed as before, while the algorithm tries to combine an item of size in $(\frac{1}{3}, 1-\Delta]$ with an item of size in $(\frac{1}{2}, \Delta]$. Even if those two items (one item of each one of the two intervals) are relatively small, still their total size is above $\frac{5}{6} \approx 0.83333$. This last algorithm was called Refined-Harmonic, and its competitive ratio is smaller than 1.636. Ramanan et al. [19] designed two algorithms called Modified Harmonic and Modified Harmonic-2. The first one has a competitive ratio below 1.61562, and it allows to combine items of many subsets with items of sizes above $\frac{1}{2}$ (and at most Δ). The second algorithm does not use only a single value of Δ, but splits the interval $(\frac{1}{2}, 1]$ further, allowing additional kinds of combinations. Its competitive ratio is approximately 1.612. For most subsets of items (where k is chosen to be in $[20, 40]$ in all these algorithms), the last two algorithms pack some proportion of the items in groups of smaller sizes, to allow it to be combined with an item of size above $\frac{1}{2}$. Intuitively, for an illustrative example, assume that $\Delta = 0.6$, and consider the items of sizes in $(\frac{1}{11}, \frac{1}{10}]$. The items that are not packed into groups of ten items should be packed into groups of four items (the parameters of the algorithms are different from those of this example). For some of the subsets the proportion is zero, and they are still packed using NF. The drawback of such algorithms (as it is exhibited by Ramanan et al. [19]) is that no matter how many thresholds there are, there can be pairs of items that can be combined into bins of optimal solutions while the algorithm does not allow it as it has fixed thresholds. Specifically, such algorithms allow to combine items of different intervals only in the case that the largest items of the two intervals fit together into a bin. This is the case with the next two harmonic type algorithms as well.

The next two papers, that of Richey [20] and that of Seiden [23] deal with a more complicated algorithm where many more subsets can be combined. The general structure is proposed in [20], and a full and corrected algorithm with its analysis is provided in [23]. For illustration, the items packed into smaller groups are called red and those packed into bins with maximum numbers of items of the subset are called blue. The goal is to combine as many bins with blue items with bins having red items as possible. Bins with red items always have small numbers of items, to allow them to be combined with relatively large items of sizes above $\frac{1}{2}$. The analysis is far from being simple, though it leads to a competitive ratio of at most 1.58889 (Heydrich and van Stee [22, 11] mention that this last value can be decreased very slightly). The analysis of [23] is based on a complicated notion called weight system. The complicated details of this analytic tool basically did not allow the research community to introduce new algorithmic methods for dealing with the online bin packing problem. We expect that our simplified analysis will not suffer from this major disadvantage.

The carefully designed subset structure eliminates many worst-case examples, but the drawback mentioned above still remains. Recently, Heydrich and van Stee [22, 11] proposed to use a method introduced by Babel et. al [1], where some items are packed based on their exact size rather than by their subset. The approach of [22, 11] which we adopt is to apply the methods of Babel et. al [1] on the largest items, of sizes in $(\frac{1}{3}, 1]$. This approach means to combine items of sizes above $\frac{1}{2}$ with items of sizes in $(\frac{1}{3}, \frac{1}{2}]$ based on their exact sizes. Moreover, the approach involves combining pairs of items of subsets of sizes contained in $(\frac{1}{3}, \frac{1}{2}]$ while keeping the smallest items of such a subset to be matched with items of sizes above $\frac{1}{2}$ (and larger items of such a subset are used to be packed into pairs), as much as possible. Prior to the work of [22, 11], all previous algorithms for classic bin packing that partition items into classes always assumed that an item of a certain subset has the maximum size when its possible packing was examined. This method simplifies the algorithm and its analysis, but it is not always a good strategy as this excludes the option of combining items that can fit together into a bin in many cases. This approach is very different from that of AF algorithms and even from NF. Moreover, an approach similar to that of Babel et. al [1] was used in an online algorithm designed in [3]. Heydrich and van Stee [22, 11] claim a competitive ratio of 1.5816 but we were not able to verify this claim.

In algorithm AH, we do not just have red and blue items, but we potentially allow several kinds of bins (that is, several and potentially a large number of colors for items of a given class, and furthermore items may change their colors once further items arrive. Due to these differences we will not use the illustration via colors of items in the description of our algorithm). For example, for the subset of items of sizes in $(\frac{1}{15}, \frac{1}{14}]$ we group items into subsets of 14 items or three items or just one item. We also use bins of the smallest items (our value of k is 43) where the total size of items is at most $\frac{17}{60}$, to allow them to be combined (among others) with items of sizes in $(\frac{1}{2}, \frac{43}{60}]$. These two features are possible due to the simple nature of our analysis, and they are crucial for getting the improved bound. Note that all items of sizes in $(0, \frac{1}{43}]$ are treated together (by the algorithm and its analysis).

In order to use just a small number of values (one or three) for each scenario that we choose by solving an auxiliary linear program, we use the concept of *containers*. A container is a set of items of one class (in the partition of potential inputs into items of similar sizes, called classes), and it can be complete if its planned number of items has arrived already or incomplete otherwise (but it is treated in the same way in both cases). Containers are of two types, where a container is either positive or negative, and a bin may contain at most one of each of them. The goal is to have as many bins as possible with both a positive and a negative container. Roughly speaking, positive containers have total sizes above $\frac{1}{2}$ and negative containers have total sizes of at most $\frac{1}{2}$. This last statement is imprecise as in most cases we consider volumes and not exact sizes, where volumes are based on the maximum sizes for the corresponding classes. There is one exception which is containers with one item of size above $\frac{1}{3}$, where the exact size is taken into account (both by the algorithm and the analysis), and it is defined to be the volume. A positive container and a negative one fit together if their total volumes does not exceed 1, and does not depend only on the classes. Our positive containers and negative containers have some relation to concepts used in [23].

In our weight based analysis, we assign weights to containers, where the number of different weights is small. Specifically, let the minimum volume of any positive container not packed with a negative container be denoted by a. We have two cases. In the simple case where all positive containers packed without negative containers have volumes of at least $\frac{2}{3}$ (i.e., $a \geq \frac{2}{3}$), we define weights as follows. Assign weights of 1 to positive containers packed without negative containers and negative containers packed without positive containers. Since we later base our weights of items on sizes, we assign these weights of 1 to all positive containers of volume at least a and all negative containers of volumes above $1-a$. We have a variable w ($0 \leq w \leq 1$) such that other positive containers have weights of w and other negative containers have weights of $1-w$. Those weights are called the required weights of containers (the actual weights can be larger but not smaller). Given the approximate proportions of items of each class packed in every type of container, we compute a weighted average (based on the containers of every item) to define weights of items using the required weights of containers. The case where $a < \frac{2}{3}$ is more interesting as a negative container with one item of size in $(\frac{1}{3}, \frac{1}{2}]$ and a positive container with one item of size above $\frac{1}{2}$ can be packed into one bin if the total size of the two items does not exceed 1 (i.e., the volumes of their containers are the exact sizes of these two items). Thus, the exact value a is crucial and not only its class, and additionally the class and even the exact value of $1-a$ play an important role. This is indeed more interesting as the analysis cannot be done for an infinite set of scenarios and thus we need to analyze intervals of a together. Here, for other classes we do the same as in the previous case, but for one class we perform a more careful analysis. This is the class containing the value $1-a$. For this class we define weights of items directly. We let the weight of an item of this class of size at most $1-a$ be a variable u, and otherwise it is a variable v, where $v \geq u$ (this class is contained in $(\frac{1}{3}, \frac{1}{2}]$). For the analysis, we found suitable values for the variables for all scenarios (this was done separately for each scenario), that is, for all possible values of a (the number of scenarios is still finite, as they are based on the dividing points of the algorithm, though not only on the classes). For every scenario where $a < \frac{2}{3}$, there are additional constraints on u, v, and w. As we do not use weights of containers in this case (for the class containing $1-a$), while the packing of pairs of items of classes contained in $(\frac{1}{3}, \frac{1}{2}]$ is performed carefully for all such classes. After selecting suitable values for those variables, all other item weights are also computed using the parameters of the algorithm.

There are also improved algorithms based on First Fit. Yao [27] designed a $\frac{5}{3}$-competitive algorithm where certain size based subsets are packed separately. Later, an algorithm of absolute competitive ratio $\frac{5}{3}$ was designed [3], which is the best possible with respect to this last measure [28] (see also [24, 7, 8]). The (asymptotic) competitive ratios should be compared to lower bounds on the competitive ratio. The current best such lower bound is 1.5403 [4] (see also [26]).

2 Algorithm AH

Notation and definitions. Similarly to previous algorithms' definitions, AH has a sequence of boundary points that are used in its precise definition: $1 = t_0 > t_1 = \frac{1}{2} > t_2 > \cdots > t_b = \frac{1}{3} > \cdots > t_M > t_{M+1} = 0$. That is 1/2 and 1/3 are always boundary points, and there is no boundary point in $(1/2, 1)$.

For every j, all items of sizes in the interval $(t_j, t_{j-1}]$ are called items of class j. We say that a class of items (and every item of this class) is *huge* if $j = 1$, it is *large* if $1 < j \le b$ (these are all items of sizes above 1/3 and at most 1/2), *small* if $b < j \le M$, and *tiny* if this is the class of items of size at most t_M (i.e., the last class which is the class of tiny items is class $M + 1$, and in general the index of a class corresponds to the index j such that t_j is the infimum size of any item of the class).

Our algorithm will pack items into containers and pack containers into bins. As the algorithm is online, a container will be packed into a bin immediately when it is created, even though it may receive additional items later. In the last case, when we say that an item is packed into a container, this means that the bin containing the container receives that item. Any container will contain items of a single class, and at most two different containers can be combined (packed) into a bin. We provide additional details on combining two containers into a bin later. Every container of items that are not tiny has a cardinality associated with it, and this is the (maximum) number of items that it is supposed to receive.

Let $\gamma_j = \lfloor \frac{1}{t_{j-1}} \rfloor$ for $j \le M$. For class j that is either large or small (but not huge or tiny, i.e., for values of j such that $2 \le j \le M$ holds), and for every i (where $1 \le i \le \gamma_j$) there is a nonnegative parameter α_{ij}, where $0 \le \alpha_{ij} \le 1$. The value α_{ij} will denote the proportion of number of containers of cardinalities i of class j items among the number of containers of class j (the term proportion corresponds to the property of the sum of proportions satisfies $\sum_i \alpha_{ij} = 1$ for all j). Such containers that will eventually receive i items of class j (unless the input terminate before this becomes possible) will be called *type i containers of class j*. That is, intuitively if we let x denote the number of containers for items of class j, we will have approximately $\alpha_{ij} \cdot x$ type i containers each of which having exactly i items of class j. For every j such that $2 \le j \le M$ and every i, we let $A_{i,j} = i \cdot t_{j-1}$. While the values α_{ij} are defined so far only for large and small classes, we see one huge item as a type 1 container. Note that the values of α_{ij} are not proportions of items but of containers for class j, and the resulting proportions of items can be computed from them (we will prove such bounds accurately later).

For classes of *large* items the notion of the cardinality of a container is slightly more delicate, and we will have exactly four possible types of containers. The first type is a *regular type 2* container (already) containing exactly two items of this class. The second type is a *declared type 2 container*, where this type consists of containers for which the algorithm already decided to pack two items of this class in the container (so the planned cardinality of the container is 2) but so far only one such item was packed into the container (one of the few next arriving items of this class, if they exist, will be packed there, in which case the type will be changed into a regular type 2 container). The third is a *regular type 1* container, where such a container has one item of the class and cannot ever have (in future steps) an additional item of this class (such a container will be already combined with a container of another class that is packed into the same bin). The fourth and last type of a container of large items is a *temporary type 1 container*. A container of this last type currently has one item of the class but sometimes it will get an additional item of this class in future steps (and in this case its type will be changed at that time to regular type 2, its type can change to declared type 2 or regular type 1 as well, but in those cases it does not happen as a result of packing a new item to this container). Given a class of large items, the number of declared type 2 containers will be at most four throughout the execution of the algorithm (as we will prove below) while the numbers of containers of type 1 (of both kinds) and containers of regular type 2 can grow unbounded as the length of the input grows, though we will show certain properties on the relations between their numbers maintained by the algorithm.

The set of the union of containers of regular type 2 and declared type 2 are called type 2 containers, and the set of the union of containers of regular type 1 and temporary type 1 are called type 1 containers. The parameters α_{1j} and α_{2j} of a large class j determine the approximate proportions of type 1 containers and type 2 containers, respectively.

For class $M+1$ (of the tiny items), instead of the definitions above, there is a sequence of p possible upper bounds on the total sizes of items packed into containers of this class: $1 \geq A_{p,M+1} > A_{p-1,M+1} > \cdots > A_{1,M+1} \geq t_M$, and we let the positive parameters $\alpha_{i,M+1} > 0$ for $i = 1, \ldots, p$ denote the proportion of numbers of containers of class $M+1$ with items of total size in the interval $(A_{i,M+1} - t_M, A_{i,M+1}]$ (this is the planned total size of items for such a container). Such containers will be called *type i containers of class $M+1$*. The values of α_{ij} for all i, j are selected heuristically via a search procedure carried out by a computer program. Any such set of parameters give a different algorithm and our proof of the numerical value of the upper bound is for one specific set of parameters that we provide explicitly.

The *volume* of a container of type i of class j is defined as follows: If $i = 1$ and $1 \leq j \leq b$ (that is, for items of sizes above $1/3$), the volume of the container is the size of its (unique) item, and otherwise ($i = 2$ and $2 \leq j \leq b$ or $i \geq 1$ and $j > b$) it is $A_{i,j}$. That is, the volume is usually simply the largest total size that the container can occupy, but for a container that contains a single large or huge item, the volume is the *exact* size of the item (there is one exception where the bin already contains one large item and it is planned to contain another item of the same class). In most cases we would like the volume of a container to be known when it is created, which is possible for containers such that their planned contents are known (in the sense that for example type i containers of a non tiny class j are planned to contain i items finally). However, for large items such containers with a single item may be temporary type 1 containers, in which case there is still no planning of contents for them. In this last case, the volume of the container is the size of its unique item. However, the volume of such a container may change in the case the algorithm will decide to pack another item of the same class (no matter if it packs that other item immediately at the time of decision or whether we decide to pack such an item later) into this container and transform it into a type 2 container. The volume of a declared type 2 container of class j is $A_{2,j} = 2 \cdot t_{j-1}$ (the volume is based on its complete contents, no matter whether they are present already or not, as it is the case for classes of small or tiny items).

We say that a container is *negative* if its volume is at most $1/2$ and otherwise it is *positive*. Obviously, two positive containers cannot be packed into one bin. We will also not pack two or more negative containers into a bin together. Thus, a bin containing two containers will contain one positive container and one negative container, and no bin will contain more than two containers.

The rules for packing containers. The algorithm AH which we define next will pack items into containers and pack containers into bins according to rules we will define. Recall that the packing of containers into bins will be such that every bin will have at most one positive container and at most one negative container. Obviously, a bin is nonempty if it has at least one container and at most two containers. We say that a nonempty bin is negative if it has a negative container and does not have a positive container, it is positive if it has a positive container and does not have a negative container, and it is neutral if it has both a negative container and a positive container.

It is unknown whether a temporary type 1 container will eventually be positive or negative. Therefore, such a container will not be combined in a bin with another container as long as its type is not changed. Moreover, it is seen as a negative container until it changes its type (so its bin is negative as long as the container is of temporary type 1). Specifically, it remains a negative container if a positive container joins it (and its bin becomes neutral), and in this case it becomes a regular type 1 container (and remains negative), and it becomes a positive container if its type changes to type 2. It can also happen that a temporary type 1 container will remain such till the termination of the input and the action of AH (and its bin remains negative). It is important to note that the difference between regular type 1 containers of a large class and temporary type 1 containers of the same class is that each of the former containers is already packed into a bin with a positive container (of some class), while the latter are not packed with other containers (in fact, the corresponding items are placed into their own bins, one item per bin).

For every class j, we denote by n_j the number of containers of class j. Let n_{ij} denote the number of containers of type i of class j. We also let N_j denote the number of items of class j at that moment. We often consider the values n_j and n_{ij} just prior to the packing of a new item, when N_j was already increased but the new item not packed yet so the values n_j and n_{ij} are not updated yet.

We say that two containers *fit together* if their total volume is at most 1. In what follows, when we refer to packing an item e - or more precisely, packing a container containing e (which was just created and therefore contains only e) into existing bins using Best Fit - we refer to packing e (or the container containing e) into the bin with a container of largest volume where the existing container and e (or the container containing e) fit together. For the original version of Best Fit, actual sizes are taken into account, but here we base this rule on volumes (as for a container with a single large or huge item the volume is equal to the size of the item, if we select one such container among a set of this last kind of containers, our action is equivalent to the standard application of Best Fit).

Packing rules of a new item. Next, we define the packing rules of the algorithm when a new item of class j arrives. The algorithm is defined for each step, based on the class of the new item.

A huge item. Recall that a huge item is immediately packed into a positive container containing only this item. Use Best Fit (applied on volumes, as explained above) to pack the created container into an existing bin, out of existing negative bins, such that the two containers (the new one with the huge item and the negative one of the negative bin) fit together. The only case where the new huge item joins a bin with a large item of some class j' is the case where the container of class j' is a temporary type 1 container, and in this case the type of this container of class j' is changed into regular type 1. If no bin can accommodate the container of the new item according to those packing rules, that is, for every negative bin, the total volume together with the new item is too big (or there is no negative bin at all), then use a new bin for the positive container of the new item (this new bin becomes a positive bin).

An item of a class of small or tiny items. For these classes we define the concept of an open container. Informally, an open container (of class j) can receive at least one additional item of class j. As a new container is introduced in order to pack an item, any container (of any type and class) already has at least one item of the corresponding class. If $b < j \leq M$, an open type i container of class j is one where the total number of the items in the container is strictly smaller than i. Once such a container receives i items, it is closed. For $j = M + 1$, a type i container of this class will be open starting the time it is created and while the total size of items in it is positive and at most $A_{i,M+1} - t_M$. Once it reaches a total size above $A_{i,M+1} - t_M$, it will be closed. For all cases of packing a small or tiny item, a new container of some class will be used only if there is no open container of the same class, and thus, in particular, there will be at most one open container for each j (and the corresponding value of i will always be one such that $\alpha_{ij} > 0$).

When a new item of class j (such that $j > b$) arrives, if there is an open container of some type i of class j, then pack the item there (there can be at most one such container, so there are no ties in this case). Otherwise, open a new container for it (the details of the type are given below). After packing the new item into the container (and packing its container into a bin if it is a new container), close the container if necessary, based on its type and the rules above.

In the case that a new container is used for the item, we define the process of packing the item in more detail. Prior to packing the item, we define the type of the new open container. As the item is not packed yet, n_j is the number of containers of class j excluding the container opened for the new item. Find the minimum value of i such that $\alpha_{ij} > 0$ and so far there are at most $\lfloor \alpha_{ij} \cdot n_j \rfloor$ type i containers of class j (i.e., $n_{ij} \leq \lfloor \alpha_{ij} \cdot n_j \rfloor$, where the values n_{ij} do not include the new container which will be opened). Such an index i exists as otherwise there are more than n_j containers of class j. More precisely, since $\sum_i \alpha_{i,j} = 1$, there is always a value of i satisfying that $\alpha_{ij} > 0$ such that so far we opened at most $\lfloor \alpha_{i,j} \cdot n_j \rfloor$ type i containers of class j. Open a new type i container of class j containing the new item (increasing both n_j and n_{ij}). Observe that this opening of a new container defines its volume as well as whether it is a positive container or a negative container.

Next, we decide where to pack this new container. First consider the case where this container is a negative container. Then, if there is a positive bin, such that the new container fits into the bin according to its volume, then use that bin to pack the new container. This last case includes the possibility that the positive container is a type 2 container of a large class (regular or declared). If there are multiple options for choosing a bin, one of them is chosen arbitrarily.

Otherwise (there is no positive bin where the new negative container can be added), the algorithm checks the option of using a bin with a temporary type 1 container of some class of large items. Assume that there is a negative bin B such that the following two conditions are satisfied. The first condition is that the bin B has a temporary type 1 container of class j' such that a positive container of class j' (with two items) will fit together with the new (negative) container. The second condition is that there are at most $\lfloor \alpha_{2j'} \cdot n_{j'} \rfloor - 1$ type 2 containers of class j' (before the packing of the new item is performed). Then, pack the new negative container into B, and define the container of class j' packed into B as a declared type 2 container. This last container of class j' will get one of the next items of class j' that will arrive, which will happen before any new container is opened for any new class j' item, see below. If there are multiple options for choosing B, one of the classes of large items is chosen arbitrarily (among those that can be used), and a temporary type 1 container of this class with maximum volume is selected, i.e., we use Best Fit in this case. This last packing step is possible as a temporary type 1 container is never packed with another container into a bin (if another container joins it, its type is changed).

Otherwise (if there is no suitable positive bin and no class of large items has a suitable temporary type 1 container that can be used under the required conditions), pack the new negative container into a new bin.

Finally, consider the case where the new container is a positive container. Then, if there is a negative bin whose container is not a temporary type 1 container, such that the new container fits together with it, then use such a bin to pack the new container. Otherwise, if there is a temporary type 1 container with one large item of a class j' where the new container fits, then pack the new positive container into this bin and define the container of class j' in this bin as a regular type 1 container. The class j' can be chosen arbitrarily if there are multiple options, and among the temporary type 1 containers of class j', one of maximum volume (out of those that can be used) is selected, i.e., once again we use Best Fit. Otherwise, pack the new positive container into a new bin.

A large item of a class j. If there is a declared type 2 container of class j, pack the item there (as a second item) and change it into a regular type 2 container (breaking ties arbitrarily). This packing rule is checked first, and we apply it whenever possible. We continue to the other cases in the situation where there is no such declared type 2 container.

If the number of type 2 containers equals $\lfloor \alpha_{2j} \cdot n_j \rfloor$ (that is, we should not increase the number of type 2 containers at this stage), then pack the new item into a new negative container. To pack the container into a bin, do as follows. If there is a positive bin where the new negative container fits, then use Best Fit to pack it as a regular type 1 container of class j (its volume is defined accordingly as the size of the new item) together with a positive container (this positive container is not of large items, as three large items cannot be packed into a bin together). Otherwise the new container is packed into a new bin, in which case it is defined to be a temporary type 1 container.

Otherwise (that is, the number of type 2 containers is strictly smaller than $\lfloor \alpha_{2j} \cdot n_j \rfloor$), we will increase the number of regular type 2 containers or the number of declared type 2 containers of this class in the current iteration as follows. If there is a negative bin B where a type 2 container of class j fits, then pack the item into a new declared type 2 container of class j and pack this container into this bin B. Otherwise, if there is a temporary type 1 container of class j, then we pack the new item using Best Fit (considering only temporary type 1 containers of class j, and selecting such a container of largest volume) and change the type of this container into a regular type 2 container. Otherwise (all containers of class j are either regular type 1 or regular type 2, we should increase the number of type 2 containers, and a new container with two items of this class cannot be packed into an existing bin), we open a new declared type 2 container for the new item and open a new bin for this declared type 2 container (and pack it there).

A sketch of the analysis. In the analysis, we see a pair of a negative container and a positive container, packed together in a bin, as matched to each other, and each one of them is seen as matched (while every container packed into a bin without another container is unmatched). Let $a' = 1 - s_{\min}/2$ where $s_{\min}$ is the smallest item size in the examined input, and let a be the smallest volume of a positive container that is unmatched, if it exists. If no unmatched positive container exists, let $a = a'$. If $a > a'$, decrease the value of a to be a'. A simple property of the algorithm is that it tries to match a positive container and a negative container whenever possible. Thus every positive container of volume smaller than a is matched and every negative container of volume at least $1 - a$ is matched.

We define a *finite* set of scenarios according to the value of a. To do that we define a set of values V as follows. $V = \{A_{i,j}, 1 - A_{i,j} : j = 2, 3, \ldots, M + 1, \alpha_{ij} > 0\} \cup \{t_1, t_2, \ldots, t_M, t_{M+1}\}$ and $V' = \{x \in V : x \leq 1/2\}$ (in particular, $\frac{1}{2} \in V'$). Note that the set V' contains (among other) all boundary points t_j (for all $j \geq 1$), even for values of j for which $\alpha_{1j} = 0$. The name of a scenario is an interval $(x, y]$ between consecutive values in V'. Using this partition, we ensure that if the scenario is $(x, y]$, then there is no $i \geq 2$ and class j such that $\alpha_{ij} > 0$ and the volume of a container of type i of class j is in (x, y) or in $(1 - y, 1 - x)$.

The first step for analyzing each scenario is to obtain a good weight function for the scenario, in the sense that the analysis will be as tight as possible and can be done using a computer assisted proof within a small running time. The weight function defines size based weights for values in $(0, 1]$. The goal is to define weights such that the cost of the algorithm is roughly the total weight of all input items (a weight function satisfying this requirement is called here *valid*), and if the target competitive ratio is R, the cost of an optimal solution is at least the total weight divided by R (this can be proved by showing that no bin can contain items of total weight above R). Then, for an input I, letting $w(I)$ denote its total weight, (and as defined above, letting $OPT(I)$ the optimal cost for I, and $A(I)$ the number of bins used by A), we will have $A(I) \leq w(I) + c$, $OPT(I) \geq \frac{w(I)}{R}$, which shows that $A(I) \leq R \cdot OPT(I) + c$. This last argument is the standard argument for weight functions based analysis [14, 15, 16, 18, 19].

In order to define a suitable function, we will solve a linear program defined below (this linear program has only four variables w, u, v and R, and in some cases it actually has only two variables w and R). More precisely, we will provide a feasible solution for this linear program that is very close to the optimal one (but we only use its feasibility and do not prove that it is almost optimal). The weights of specific sizes will be based on the values w, u, v (or just on w, if the others are undefined), and on some of the parameters of the algorithm (the α_{ij} values for the given class).

We define a quantity for each container called the *required weight* of the container, and its goal is to introduce a uniform value such that weights of items are defined based on these values, in order to satisfy all requirements. This quantity is defined for a class that is not the threshold class or is not a large class. If the threshold class k (the class containing $1 - a$) is a large class, we keep this quantity undefined for that class. For a positive container of volume at least a, the required weight of the container is 1. For a positive container of volume in the interval $(1/2, a)$, the required weight of the container is denoted as w. This will be a decision variable of the forthcoming linear program. The required weight of a negative container is 1 if its volume is larger than $1 - a$ and otherwise its required weight is $1 - w$. We ensure that the required weight of a container depends only on the index of the scenario $(x, y]$ and not the specific value of a in the interval $[1 - y, 1 - x)$ and there are only few exceptions that are handled separately.

The weight of a huge item is 1 if its size is at least a and w otherwise. The weight of an item of class $j \leq M$ such that either $j \neq k$ or $j > b$ is the ratio between the average required weight of a container of class j and the average number of items in a container of class j. The weight of a tiny item of size s is s times the ratio between the average required weight of a container of tiny items and the average (lower bounds on the) total size of items in a container of tiny items. The weight of items of class $j = k$ that is a large class is as follows. An item of this class has weight u if its size is at most $1 - a$ and otherwise a weight of v. We find linear inequalities on the variables u, v, w that ensure that the resulting weight function is valid. By solving a linear program we can find such values of u, v, w that minimize the corresponding competitive ratio that can be proven using this weight function. In this linear program the goal is to minimize R that is an upper bound on the total weight of items that can fit into one bin subject to the additional constraints on u, v, w ensuring that the resulting weight function is indeed valid.

In this way we get a table showing for each scenario the set of the values of u, v, w (or only w for scenarios where the threshold class is not large) that define the weight function that we use for the scenario. Using these weight functions we show the correctness of our main result, namely that the competitive ratio of AH is at most 1.57828956.

References

1 L. Babel, B. Chen, H. Kellerer, and V. Kotov. Algorithms for on-line bin-packing problems with cardinality constraints. *Discrete Applied Mathematics*, 143(1-3):238–251, 2004.

2 B. S. Baker and E. G. Coffman, Jr. A tight asymptotic bound for next-fit-decreasing bin-packing. *SIAM J. on Algebraic and Discrete Methods*, 2(2):147–152, 1981.

3 J. Balogh, J. Békési, Gy. Dósa, J. Sgall, and R. van Stee. The optimal absolute ratio for online bin packing. In *Proc. of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA2015)*, pages 1425–1438, 2015.

4 J. Balogh, J. Békési, and G. Galambos. New lower bounds for certain bin packing algorithms. *Theoretical Computer Science*, 1:1–13, 2012.

5 E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum, editor, *Approximation algorithms.* PWS Publishing Company, 1997.

6 J. Csirik and G. J. Woeginger. On-line packing and covering problems. In *A. Fiat and G. J. Woeginger, editors,* Online Algorithms: The State of the Art, pages 147–177, 1998.

7 Gy. Dósa and J. Sgall. First Fit bin packing: A tight analysis. In *Proc. of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS2013)*, pages 538–549, 2013.

8 Gy. Dósa and J. Sgall. Optimal analysis of Best Fit bin packing. In *The 41st International Colloquium on Automata, Languages and Programming (ICALP2014)*, pages 429–441, 2014.

9 L. Epstein and A. Levin. A robust APTAS for the classical bin packing problem. *Mathematical Programming*, 119(1):33–49, 2009.

10 W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1+\varepsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

11 S. Heydrich and R. van Stee. Beating the harmonic lower bound for online bin packing. In *Proc. of 43rd International Colloquium on Automata, Languages, and Programming (ICALP2016)*, pages 41:1–41:14, 2016.

12 R. Hoberg and T. Rothvoss. A logarithmic additive integrality gap for bin packing. In *Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA2017)*, pages 2616–2625, 2017.

13 K. Jansen and K.-M. Klein. A robust AFPTAS for online bin packing with polynomial migration. In *Proc. of the 40th International Colloquium on Automata, Languages, and Programming (ICALP2013), part I*, pages 589–600, 2013.

14 D. S. Johnson. *Near-optimal bin packing algorithms.* PhD thesis, MIT, Cambridge, MA, 1973.

15 D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 1974.

16 D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3:256–278, 1974.

17 N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proc. of the 23rd Annual Symposium on Foundations of Computer Science (FOCS'82)*, pages 312–320, 1982.

18 C. C. Lee and D. T. Lee. A simple online bin packing algorithm. *Journal of the ACM*, 32(3):562–572, 1985.

19 P. Ramanan, D. J. Brown, C. C. Lee, and D. T. Lee. Online bin packing in linear time. *Journal of Algorithms*, 10:305–326, 1989.

20 M. B. Richey. Improved bounds for harmonic-based bin packing algorithms. *Discrete Applied Mathematics*, 34(1–3):203–227, 1991.

21 T. Rothvoss. Better bin packing approximations via discrepancy theory. *SIAM Journal on Computing*, 45(3):930–946, 2016.

22 R. van Stee S. Heydrich. Beating the harmonic lower bound for online bin packing. *The Computing Res. Rep. (CoRR)*, abs/1707.01728, 2017. `arXiv:1511.00876v3`.

23 S. S. Seiden. On the online bin packing problem. *Journal of the ACM*, 49(5):640–671, 2002.

24 D. Simchi-Levi. New worst-case results for the bin-packing problem. *Naval Research Logistics*, 41(4):579–585, 1994.

25 J. D. Ullman. The performance of a memory allocation algorithm. Technical Report 100, Princeton University, Princeton, NJ, 1971.

26 A. van Vliet. An improved lower bound for online bin packing algorithms. *Information Processing Letters*, 43(5):277–284, 1992.

27 A. C. C. Yao. New algorithms for bin packing. *Journal of the ACM*, 27:207–227, 1980.

28 G. Zhang. Private communication.

Practical Access to Dynamic Programming on Tree Decompositions

Max Bannach
Institute for Theoretical Computer Science, Universität zu Lübeck, Lübeck, Germany
bannach@tcs.uni-luebeck.de
https://orcid.org/0000-0002-6475-5512

Sebastian Berndt
Department of Computer Science, Kiel University, Kiel, Germany
seb@informatik.uni-kiel.de
https://orcid.org/0000-0003-4177-8081

Abstract

Parameterized complexity theory has lead to a wide range of algorithmic breakthroughs within the last decades, but the practicability of these methods for real-world problems is still not well understood. We investigate the practicability of one of the fundamental approaches of this field: dynamic programming on tree decompositions. Indisputably, this is a key technique in parameterized algorithms and modern algorithm design. Despite the enormous impact of this approach in theory, it still has very little influence on practical implementations. The reasons for this phenomenon are manifold. One of them is the simple fact that such an implementation requires a long chain of non-trivial tasks (as computing the decomposition, preparing it,...). We provide an easy way to implement such dynamic programs that only requires the definition of the update rules. With this interface, dynamic programs for various problems, such as 3-COLORING, can be implemented easily in about 100 lines of structured Java code.

The theoretical foundation of the success of dynamic programming on tree decompositions is well understood due to Courcelle's celebrated theorem, which states that every MSO-definable problem can be efficiently solved if a tree decomposition of small width is given. We seek to provide practical access to this theorem as well, by presenting a lightweight model-checker for a small fragment of MSO. This fragment is powerful enough to describe many natural problems, and our model-checker turns out to be very competitive against similar state-of-the-art tools.

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

Keywords and phrases fixed-parameter tractability, treewidth, model-checking

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.6

1 Introduction

Parameterized algorithms aim to solve intractable problems on instances where some parameter tied to the complexity of the instance is small. This line of research has seen enormous growth in the last decades and produced a wide range of algorithms [9]. More formally, a problem is *fixed-parameter tractable* (in FPT), if every instance I can be solved in time $f(\kappa(I)) \cdot \mathrm{poly}(|I|)$ for a computable function f, where $\kappa(I)$ is the *parameter* of I. While the impact of parameterized complexity to the theory of algorithms and complexity cannot be overstated, its practical component is much less understood. Very recently, the investigation of the practicability of fixed-parameter tractable algorithms for real-world problems has started to become an important subfield (see e.g. [18, 11]). We investigate the practicability of dynamic programming on tree decompositions – one of the most fundamental techniques of parameterized algorithms. A general result explaining the usefulness of tree decompositions

© Max Bannach and Sebastian Berndt;
licensed under Creative Commons License CC-BY

was given by Courcelle in [8], who showed that *every* property that can be expressed in monadic second-order logic is fixed-parameter tractable if it is parameterized by tree width. By combining this result (known as Courcelle's Theorem) with the $f(\mathrm{tw}(G)) \cdot |G|$ algorithm of Bodlaender [7] to compute an optimal tree decomposition in FPT-time, a wide range of graph-theoretic problems is known to be solvable on these tree-like graphs. Unfortunately, both ingredients of this approach are very expensive in practice.

One of the major achievements concerning practical parameterized algorithms was the discovery of a practically fast algorithm for treewidth due to Tamaki [19]. Concerning Courcelle's Theorem, there are currently two contenders concerning efficient implementations of it: D-Flat, an Answer Set Programming (ASP) solver for problems on tree decompositions [1]; and Sequoia, an MSO solver based on model checking games [17]. Both solvers allow to solve very general problems and the corresponding overhead might, thus, be large compared to a straightforward implementation of the dynamic programs for specific problems.

Our Contributions. In order to study the practicability of dynamic programs on tree decompositions, we expand our tree decomposition library Jdrasil with an easy to use interface for such programs: The user only needs to specify the *update rules* for the different kind of nodes within the tree decomposition. The remaining work – computing a suitable optimized tree decomposition and performing the actual run of the dynamic program – are done by Jdrasil. This allows users to implement a wide range of algorithms within very few lines of code and, thus, gives the opportunity to test the practicability of these algorithms quickly. This interface is presented in Section 3.

While D-Flat and Sequoia solve very general problems, the experimental results of Section 5 show that naïve implementations of dynamic programs might be much more efficient. In order to balance the generality of MSO solvers and the speed of direct implementations, we introduce a small MSO fragment, that avoids quantifier alternation, in Section 4. By concentrating on this fragment, we are able to build a model-checker, called Jatatosk, that runs nearly as fast as direct implementations of the dynamic programs. To show the feasibility of our approach, we compare the running times of D-Flat, Sequoia, and Jatatosk for various problems. It turns out that Jatatosk is competitive against the other solvers and, furthermore, its behaviour is much more consistent (i. e. it does not fluctuate greatly on similar instances). We conclude that concentrating on a small fragment of MSO gives rise to practically fast solvers, which are still able to solve a large class of problems on graphs of bounded treewidth.

2 Preliminaries

All graphs considered in this paper are undirected, that is, they consists of a set of vertices V and of a symmetric edge-relation $E \subseteq V \times V$. We assume the reader to be familiar with basic graph theoretic terminology, see for instance [10]. A *tree decomposition* of a graph $G = (V, E)$ is a tuple (T, ι) consisting of a rooted tree T and a mapping ι from nodes of T to sets of vertices of G (which we call *bags*) such that (1) for all $v \in V$ there is a node n in T with $v \in \iota(n)$, (2) for every edge $\{v, w\} \in E$ there is a node m in T with $\{v, w\} \subseteq \iota(m)$, and (3) the set $\{\, x \mid v \in \iota(x) \,\}$ is connected in T for every $v \in V$. The *width* of a tree decomposition is the maximum size of one of its bags minus one, and the *treewidth* of G, denoted by $\mathrm{tw}(G)$, is the minimum width any tree decomposition of G must have.

In order to describe dynamic programs over tree decompositions, it turns out be helpful to transform a tree decomposition into a more structured one. A *nice tree decomposition* is a triple (T, ι, η) where (T, ι) is a tree decomposition and $\eta\colon V(T) \to \{\mathrm{leaf}, \mathrm{introduce},$

forget, join} is a labeling such that (1) nodes labeled "leaf" are exactly the leaves of T, and the bags of these nodes are empty; (2) nodes n labeled "introduce" or "forget" have exactly one child m such that there is exactly one vertex $v \in V(G)$ with either $v \notin \iota(m)$ and $\iota(n) = \iota(m) \cup \{v\}$ or $v \in \iota(m)$ and $\iota(n) = \iota(m) \setminus \{v\}$, respectively; (3) nodes n labeled "join" have exactly two children x, y with $\iota(n) = \iota(x) = \iota(y)$. A *very nice tree decomposition* is a nice tree decomposition that also has exactly one node labeled "edge" for every $e \in E(G)$, which virtually introduces the edge e to the bag – i. e., whenever we introduce a vertex, we assume it to be "isolated" in the bag until its incident edges are introduced. It is well known that any tree decomposition can efficiently be transformed into a very nice one without increasing its width (essentially traverse through the tree and "pull apart" bags) [9]. Whenever we talk about tree decompositions in the rest of the paper, we actually mean very nice tree decompositions. However, we want to stress out that all our interfaces also support "just" nice tree decompositions.

We assume the reader to be familiar with basic logic terminology and give just a brief overview over the syntax and semantic of monadic second-order logic (MSO), see for instance [13] for a detailed introduction. A *vocabulary* (or *signature*) $\tau = (R_1^{a_1}, \dots, R_n^{a_n})$ is a set of *relational symbols* R_i of arity $a_i \geq 1$. A τ*-structure* is a set U – called *universe* – together with an *interpretation* $R_i^U \subseteq R^{a_i}$ of the relational symbols. Let $x_1, x_2, \dots$ be a sequence of *first-order variables* and $X_1, X_2, \dots$ be a sequence of *second-order variables* X_i of arity $\mathrm{ar}(X_i)$. The atomic τ-formulas are $x_i = x_j$ for two first-order variables and $R(x_{i_1}, \dots, x_{i_k})$, where R is either a relational symbol or a second-order variable of arity k. The set of τ-formulas is inductively defined by (1) the set of atomic τ-formulas; (2) Boolean connections $\neg\phi$, $(\phi \vee \psi)$, and $(\phi \wedge \psi)$ of τ-formulas ϕ and ψ; (3) quantified formulas $\exists x\phi$ and $\forall x\phi$ for a first-order variable x and a τ-formula ϕ; (4) quantified formulas $\exists X\phi$ and $\forall X\phi$ for a second-order variable X of arity 1 and a τ-formula ϕ. The set of *free variables* of a formula ϕ consists of the variables that appear in ϕ but are not bounded by a quantifier. We denote a formula ϕ with free variables $x_1, \dots, x_k, X_1, \dots, X_\ell$ as $\phi(x_1, \dots, x_k, X_1, \dots, X_\ell)$. Finally, we say a τ-structure $\mathcal{S}$ with an universe U is a *model* of an τ-formula $\phi(x_1, \dots, x_k, X_1, \dots, X_\ell)$ if there are elements $u_1, \dots, u_k \in U$ and relations $U_1, \dots, U_\ell$ with $U_i \subseteq U^{\mathrm{ar}(X_i)}$ with $\phi(u_1, \dots, u_k, U_1, \dots, U_\ell)$ being true in $\mathcal{S}$. We write $\mathcal{S} \models \phi(u_1, \dots, u_k, U_1, \dots, U_\ell)$ in this case.

▶ **Example 1.** Graphs can be modeled as $\{E^2\}$-structures with a symmetric interpretation of E. Properties such as "is 3-colorable" can then be described by formulas as:

$$\tilde{\phi}_{3\mathrm{col}} = \exists R \exists G \exists B\ (\forall x\, R(x) \vee G(x) \vee B(x)) \wedge (\forall x \forall y\, E(x,y) \rightarrow \bigwedge_{C \in \{R,G,B\}} \neg C(x) \vee \neg C(y)).$$

For instance, we have $\models \tilde{\phi}_{3\mathrm{col}}$ and $\not\models \tilde{\phi}_{3\mathrm{col}}$. We write $\tilde{\phi}$ whenever a more refined version of ϕ will be given later on.

The *model-checking* problem asks, given a logical structure $\mathcal{S}$ and a formula ϕ, if $\mathcal{S} \models \phi$ holds. A *model-checker* is a program that solves this problem and outputs an assignment to its free and bounded variables if $\mathcal{S} \models \phi$ holds.

3 An Interface for Dynamic Programming on Tree Decompositions

It will be convenient to recall a classical viewpoint of dynamic programming on tree decompositions to illustrate why our interface is designed the way it is. We will do so by the guiding example of 3-COLORING: Is it possible to color vertices of a given graph with three colors such that adjacent vertices never share the same color? Intuitively, a dynamic program for 3-COLORING will work bottom-up on a very nice tree decomposition and manages a set of possible colorings per node. Whenever a vertex is introduced, the program "guesses" a color for this vertex; if a vertex is forgotten we have to remove it from the bag and identify configurations that become eventually equal; for join bags we just have to take the configurations that are present in both children; and for edge bags we have to reject colorings in which both endpoints of the introduced edge have the same color. To formalize this vague algorithmic description, we view it from the perspective of automata theory.

3.1 The Tree Automaton Perspective

Classically, dynamic programs on tree decompositions are described in terms of tree automata [13]. Recall that in a very nice tree decomposition the tree T is rooted and binary; we assume that the children of T are ordered. The mapping ι can then be seen as a function that maps the nodes of T to symbols from some alphabet Σ. A naïve approach to manage ι would yield a huge alphabet (depending on the size of the graph). We thus define the so called *tree-index*, which is a map $\mathrm{idx}\colon V(G) \to \{0, \dots, \mathrm{tw}(G)\}$ such that no two vertices that appear in the same bag share a common tree-index. The existence of such an index follows directly from the property that every vertex is forgotten exactly once: We can simply traverse T from the root to the leaves and assign a free index to a vertex V when it is forgotten, and release the used index once we reach an introduce bag for v. The symbols of Σ then only contain the information for which tree-index there is a vertex in the bag. From a theoreticians perspective this means that $|\Sigma|$ depends only on the treewidth; from a programmers perspective the tree-index makes it much easier to manage data structures that are used by the dynamic program.

▶ **Definition 2** (Tree Automaton). A nondeterministic bottom-up *tree automaton* is a tuple $A = (Q, \Sigma, \Delta, F)$ where Q is a set of *states* with a subset $F \subseteq Q$ of *accepting states*, Σ is an *alphabet*, and $\Delta \subseteq (Q \cup \{\bot\}) \times (Q \cup \{\bot\}) \times \Sigma \times Q$ is a *transition relation* in which $\bot \notin Q$ is a special symbol to treat nodes with less than two children. The automaton is *deterministic* if for every $x, y \in Q \cup \{\bot\}$ and every $\sigma \in \Sigma$ there is exactly one $q \in Q$ with $(x, y, \sigma, q) \in \Delta$.

▶ **Definition 3** (Computation of a Tree Automaton). The *computation of a tree automaton* $A = (Q, \Sigma, \Delta, F)$ on a labeled tree (T, ι) with $\iota\colon V(T) \to \Sigma$ and root $r \in V(T)$ is an assignment $q\colon V(T) \to Q$ such that for all $n \in V(T)$ we have (1) $(q(x), q(y), \iota(n), q(n)) \in \Delta$ if n has two children x, y; (2) $(q(x), \bot, \iota(n), q(n)) \in \Delta$ or $(\bot, q(x), \iota(n), q(n)) \in \Delta$ if n has one child x; (3) $(\bot, \bot, \iota(n), q(n)) \in \Delta$ if n is a leaf. The computation is *accepting* if $q(r) \in F$.

Simulating Tree Automata. A dynamic program for a decision problem can be formulated as a nondeterministic tree automaton that works on the decomposition, see the left side of Figure 1 for a detailed example. Observe that a nondeterministic tree automaton A will process a labeled tree (T, ι) with n nodes in time $O(n)$. When we simulate such an automaton deterministically, one might think that a running time of the form $O(|Q| \cdot n)$ is sufficient, as the automaton could be in any potential subset of the Q states at some node of the tree. However, there is a pitfall: For every node we have to compute the set of potential states of the automaton depending on the sets of potential states of the children of that node, leading to a quadratic dependency on $|Q|$. This can be avoided for transitions of the form $(\bot, \bot, \iota(x), p)$, $(q, \bot, \iota(x), p)$, and $(\bot, q, \iota(x), p)$, as we can collect potential successors of every state of the child and compute the new set of states in linear time with respect to the cardinality of the set. However, transitions of the form $(q_i, q_j, \iota(x), p)$ are difficult, as we now have to merge two sets of states. In detail, let x be a node with children y and z and let

Q_y and Q_z be the set of potential states in which the automaton eventually is in at these nodes. To determine Q_x we have to check for every $q_i \in Q_y$ and every $q_j \in Q_z$ if there is a $p \in Q$ such that $(q_i, q_j, \iota(x), p)$. Note that the number of states $|Q|$ can be quite large even for moderately sized parameters k, as $|Q|$ is typically of size $2^{\Omega(k)}$, and we will thus try to avoid this quadratic blow-up.

▶ **Observation 4.** *A tree automaton can be simulated in time* $O(|Q|^2 \cdot n)$.

Unfortunately, the quadratic factor in the simulation cannot be avoided in general, as the automaton may very well contain a transition for all possible pairs of states. However, there are some special cases in which we can circumnavigate the increase in the running time.

▶ **Definition 5** (Symmetric Tree Automaton)**.** A *symmetric* nondeterministic bottom-up tree automaton is a nondeterministic bottom-up tree automaton $A = (Q, \Sigma, \Delta, F)$ in which all transitions $(l, r, \sigma, q) \in \Delta$ satisfy either $l = \perp$, $r = \perp$, or $l = r$.

Assume as before that we wish to compute the set of potential states for a node x with children y and z. Observe that in a symmetric tree automaton it is sufficient to consider the set $Q_y \cap Q_z$ and that the intersection of two sets can be computed in linear time if we take some care in the design of the underlying data structures.

▶ **Observation 6.** *A symmetric tree automaton can be simulated in time* $O(|Q| \cdot n)$.

The right side of Figure 1 illustrates the deterministic simulation of a symmetric tree automaton. The massive time difference in the simulation of tree automata and symmetric tree automata significantly influenced the design of the algorithms in Section 4, in which we try to construct an automaton that is 1) "as symmetric as possible" and 2) allows to take advantage of the "symmetric parts" even if the automaton is not completely symmetric.

3.2 The Interface

We introduce a simple Java-interface to our library Jdrasil, which originally was developed for the computation of tree decompositions only. The interface is build up from two classes: `StateVectorFactory` and `StateVector`. The only job of the factory is to generate `StateVector` objects for the leaves of the tree decomposition, or with the terms of the previous section: "to define the initial states of the tree automaton". The `StateVector` class is meant to model a vector of potential states in which the nondeterministic tree automaton is at a specific node of the tree decomposition. Our interface does not define at all what a "state" is, or how a collection of states is managed (although most of the times, it will be a set). The only thing the interface requests a user to implement is the behaviour of the tree automaton when it reaches a node of the tree-decomposition, i. e., given a `StateVector` (for some unknown node in the tree decomposition) and the information that the automaton reaches a certain node, how does the `StateVector` for this node look like? To this end, the interface contains the methods shown in Listing 1.

Listing 1 The four methods of the interface describe the behaviour of the tree automaton. Here "T" is a generic type for vertices. Each function obtains as parameter the current bag and a tree-index "idx". Other parameters correspond to bag-type specifics, e. g. the introduced or forgotten vertex v.

```
StateVector<T> introduce(Bag<T> b, T v, Map<T, Integer> idx);
StateVector<T> forget(Bag<T> b, T v, Map<T, Integer> idx);
StateVector<T> join(Bag<T> b, StateVector<T> o, Map<T, Integer> idx);
StateVector<T> edge(Bag<T> b, T v, T w, Map<T, Integer> idx);
```

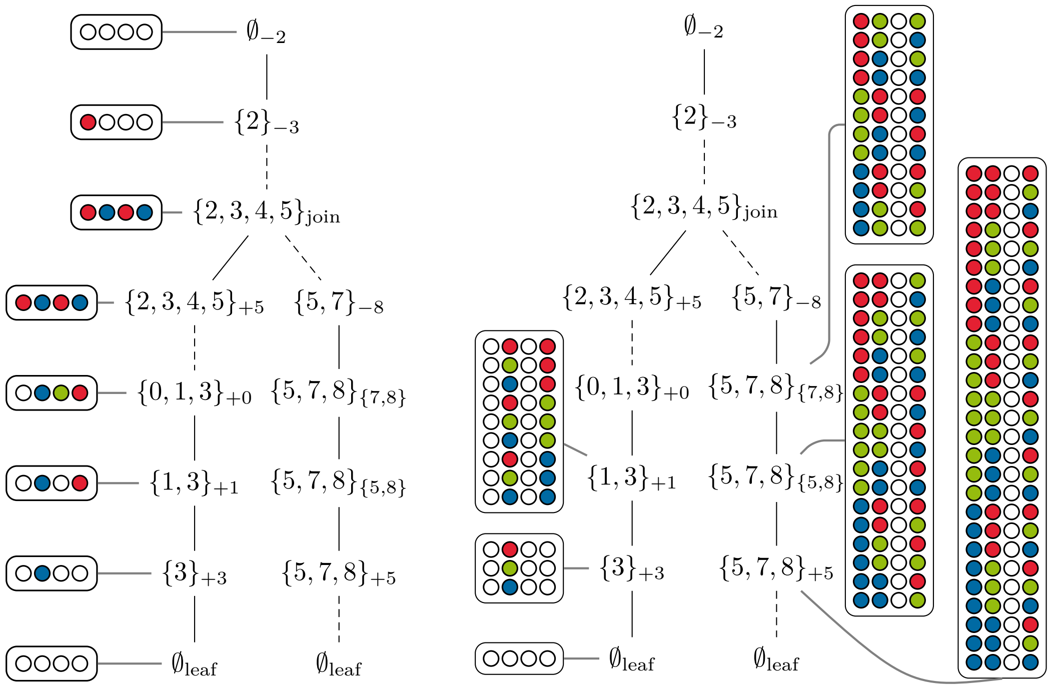


Figure 1 The *left* picture shows a part of a tree decomposition of the grid graph with vertices $\{0,\dots,9\}$. The index of a bag shows the type of the bag: a positive sign means "introduce", a negative one "forget", a pair represents an "edge"-bag, and text is self explanatory. Solid lines represent real edges of the decomposition, while dashed lines illustrate a path (i. e., there are some bags skipped). On the left branch of the decomposition a run of a nondeterministic tree automaton with tree-index $\begin{pmatrix}0&1&2&3&4&5&6&7&8\\2&3&0&1&2&3&0&1&0\end{pmatrix}$ for 3-COLORING is illustrated. To increase readability, states of the automaton are connected to the corresponding bags with gray lines, and for some nodes the states are omitted. In the *right* picture the same automaton is simulated deterministically.

This already rounds up the description of the interface, everything else is done by Jdrasil. In detail, given a graph and an implementation of the interface, Jdrasil will compute a tree decomposition[1], transform this decomposition into a very nice tree decomposition, potentially optimize the tree decomposition for the following dynamic program, and finally traverse through the tree decomposition and simulate the tree automaton described by the implementation of the interface. The result of this procedure is the `StateVector` object assigned to the root of the tree decomposition.

3.3 Example: 3-Coloring

Let us illustrate the usage of the interface with our running example of 3-COLORING. A `State` of the automaton can be modeled as a simple integer array that stores a color (an integer) for every vertex in the bag. A `StateVector` stores a set of `State` objects, i. e., essentially a set of integer arrays. Introducing a vertex v to a `StateVector` therefore means that three duplicates of each stored state have to be created, and for every duplicate a different color has to be assigned to v. Listing 2 illustrates how this operation could be realized in Java.

[1] See [6] for the concrete algorithms used by Jdrasil.

Listing 2 Exemplary implementation of the `introduce` method for 3-COLORING.

```
StateVector<T> introduce(Bag<T> b, T v, Map<T, Integer> idx) {
  Set<State> newStates = new HashSet<>();
  for (State state : states) { // 'states' is the set of states
    for (int color = 1; color <= 3; color++) {
      State newState = new State(state); // copy the state
      newState.colors[idx.get(v)] = color;
      newStates.add(newState);
    }
  }
  states = newStates;
  return this;
}
```

The three other methods can be implemented in a very similar fashion: in the `forget`-method we set the color of v to 0; in the `edge`-method we remove states in which both endpoints of the edge have the same color; and in the `join`-method we compute the intersection of the state sets of both `StateVector` objects. Note that when we forget a vertex v, multiple states may become identical, which is handled here by the implementation of the Java `Set`-class, which takes care of duplicates automatically.

A reference implementation of this 3-COLORING solver is publicly available [4], and a detailed description of it can be found in the manual of Jdrasil [5]. Note that this implementation is only meant to illustrate the interface and that we did not make any effort to optimize it. Nevertheless, this very simple implementation (the part of the program that is responsible for the dynamic program only contains about 120 lines of structured Java-code) performs surprisingly well, as the experiments in Section 5 indicate.

4 A Lightweight Model-Checker for a Small MSO-Fragment

Experiments with the coloring solver of the previous section have shown a huge difference in the performance of general solvers as D-Flat and Sequoia against a concrete implementation of a tree automaton for a specific problem (see Section 5). This is not necessarily surprising, as a general solver needs to keep track of way more information. In fact, a MSO-model-checker can probably (unless $\mathrm{P} = \mathrm{NP}$) not run in time $f(|\phi|+\mathrm{tw})\cdot\mathrm{poly}(n)$ for any elementary function f [14]. On the other hand, it is not clear (in general) what the concrete running time of such a solver is for a concrete formula or problem (see e. g. [16] for a sophisticated analysis of some running times in Sequoia). We seek to close this gap between (slow) general solvers and (fast) concrete algorithms. Our approach is to concentrate only on a small fragment of MSO, which is powerful enough to express many natural problems, but which is restricted enough to allow model-checking in time that matches or is close to the running time of a concrete algorithm for the problem. As a bonus, we will be able to derive upper bounds on the running time of the model-checker directly from the syntax of the input formula.

Based on the interface of Jdrasil, we have implemented a publicly available prototype called Jatatosk [3]. In Section 5, we perform various experiments on different problems on multiple sets of graphs. It turns out that Jatatosk is competitive against the state-of-the-art solvers D-Flat and Sequoia. Arguably these two programs solve a more general problem and a direct comparison is not entirely fair. However, the experiments do reveal that it seems very promising to focus on smaller fragments of MSO (or perhaps any other description language) in the design of treewidth based solvers.

4.1 Description of the Fragment

We only consider vocabularies τ that contain the binary relation E^2, and we only consider τ-structures with a symmetric interpretation of E^2, i. e., we only consider structures that contain an undirected graph (but may also contain further relations). The fragment of MSO that we consider is constituted by formulas of the form $\phi = \exists X_1 \dots \exists X_k \bigwedge_{i=1}^{n} \psi_i$, where the X_j are second-order variables and the ψ_i are first-order formulas of the form

$$\begin{aligned}\psi_i \in \{\, &\forall x \forall y\, E(x,y) \rightarrow \chi_i,\ \forall x \exists y\, E(x,y) \wedge \chi_i,\ \exists x \forall y\, E(x,y) \rightarrow \chi_i,\\ &\exists x \exists y\, E(x,y) \wedge \chi_i,\ \forall x\, \chi_i,\ \exists x\, \chi_i \,\}.\end{aligned}$$

Here, the χ_i are quantifier-free first-order formulas in canonical normal form. It is easy to see that this fragment is already powerful enough to encode many classical problems as 3-COLORING ($\tilde{\phi}_{\text{3col}}$ from the introduction is part of the fragment), or VERTEX-COVER (we will discuss how to handle optimization in Section 4.4): $\tilde{\phi}_{\text{vc}} = \exists S \forall x \forall y\, E(x,y) \rightarrow S(x) \vee S(y)$.

4.2 A Syntactic Extension of the Fragment

Many interesting properties, such as connectivity, can easily be expressed in MSO, but not directly in the fragment that we study. Nevertheless, a lot of these properties can directly be checked by a model-checker if it "knows" what kind of properties it actually checks. We present a *syntactic extension* of our MSO-fragment which captures such properties. The extension consist of three new second order quantifiers that can be used instead of $\exists X_i$.

The first extension is a *partition quantifier*, which quantifies over partitions of the universe:

$$\exists^{\text{partition}} X_1, \dots, X_k \equiv \exists X_1 \exists X_2 \dots \exists X_k \big(\forall x \bigvee_{i=1}^{k} X_i(x)\big) \wedge \big(\forall x \bigwedge_{i=1}^{k} \bigwedge_{j \neq i} \neg X_i(x) \wedge \neg X_j(x)\big).$$

This quantifier has two advantages. First, formulas like $\tilde{\phi}_{\text{3col}}$ can be simplified to

$$\phi_{\text{3col}} = \exists^{\text{partition}} R, G, B\ \forall x \forall y\, E(x,y) \rightarrow \bigwedge_{C \in \{R,G,B\}} \neg C(x) \vee \neg C(y),$$

and second, the model-checking problem for them can be solved more efficiently: the solver directly "knows" that a vertex must be added to exactly one of the sets.

We further introduce two quantifiers that work with respect to the symmetric relation E^2 (recall that we only consider structures that contain such a relation). The $\exists^{\text{connected}} X$ quantifier guesses an $X \subseteq U$ that is connected with respect to E (in graph theoretic terms), i. e., it quantifies over connected subgraphs. The $\exists^{\text{forest}} F$ quantifier guesses a $F \subseteq U$ that is acyclic with respect to E (again in graph theoretic terms), i. e., it quantifies over subgraphs that are forests. These quantifiers are quite powerful and allow, for instance, to express that the graph induced by E^2 contains a triangle as minor:

$$\begin{aligned}\phi_{\text{triangle-minor}} = &\exists^{\text{connected}} R\, \exists^{\text{connected}} G\, \exists^{\text{connected}} B\,.\\ &\big(\forall x\, (\neg R(x) \vee \neg G(x)) \wedge (\neg G(x) \vee \neg B(x)) \wedge (\neg B(x) \vee \neg R(x))\big)\\ \wedge\, &\big(\exists x \exists y\, E(x,y) \wedge R(x) \wedge G(y)\big) \wedge \big(\exists x \exists y\, E(x,y) \wedge G(x) \wedge B(y)\big)\\ \wedge\, &\big(\exists x \exists y\, E(x,y) \wedge B(x) \wedge R(y)\big).\end{aligned}$$

We can also express problems that usually require more involved formulas in a very natural way. For instance, the FEEDBACK-VERTEX-SET problem can be described by the following formula (again, optimization will be handled in Section 4.4): $\tilde{\phi}_{\text{fvs}} = \exists S\, \exists^{\text{forest}} F\, \forall x\, S(x) \vee F(x)$.

4.3 Description of the Model-Checker

We describe our model-checker in terms of a nondeterministic tree automaton that works on a tree decomposition of the graph induced by E^2 (note that, in contrast to other approaches in the literature, we do not work on the Gaifman graph). We define any state of the automaton as bit-vector, and we stipulate that the initial state at every leaf is the zero-vector. For any quantifier or subformula, there will be some area in the bit-vector reserved for that quantifier or subformula and we describe how state transitions effect these bits. The "algorithmic idea" behind the implementation of these transitions is not new, and a reader familiar with folklore dynamic programs on tree decompositions (for instance for VERTEX-COVER or STEINER-TREE) will probably recognize them. An overview over common techniques can be found in the standard textbooks [9, 13].

The Partition Quantifier. We start with a detailed description of the partition quantifier $\exists^{\text{partition}} X_1, \dots, X_q$ (we do not implement an additional $\exists X$ quantifier, as we can easily state $\exists X \equiv \exists^{\text{partition}} X, \bar{X}$): Let k be the maximum bag-size of the tree decomposition. We reserve $k \cdot \log_2 q$ bit in the state description, where each block of length $\log_2 q$ indicates in which set X_i the corresponding element of the bag is. On an introduce-bag (e. g. for $v \in U$), the nondeterministic automaton guesses an index $i \in \{1, \dots, q\}$ and sets the $\log_2 q$ bits that are associated with the tree-index of v to i. Equivalently, the corresponding bits are cleared when the automaton reaches a forget-bag. As the partition is independent of any edges, an edge-bag does not change any of the bits reserved for the partition quantifier. Finally, on join-bags we may only join states that are identical on the bits describing the partition (as otherwise the vertices of the bag would be in different partitions) – meaning this transition is symmetric with respect to these bits (in terms of Section 3.1).

The Connected Quantifier. The next quantifier we describe is $\exists^{\text{connected}} X$ which has to overcome the difficulty that an introduced vertex may not be connected to the rest of the bag in the moment it got introduced, but may be connected to it when further vertices "arrive". The solution to this dilemma is to manage a partition of the bag into $k' \leq k$ connected components $P_1, \dots, P_{k'}$, for which we reserve $k \cdot \log_2 k$ bit in the state description. Whenever a vertex v is introduced, the automaton either guesses that it is not contained in X and clears the corresponding bits, or it guesses that $v \in X$ and assigns some P_i to v. Since v is isolated in the bag in the moment of its introduction (recall that we work on a very nice tree decomposition), it requires its own component and is therefore assigned to the smallest empty partition P_i. When a vertex v is forgotten, there are four possible scenarios: 1) $v \notin X$, then the corresponding bits are already cleared and nothing happens; 2) $v \in X$ and $v \in P_i$ with $|P_i| > 1$, then v is just removed and the corresponding bits are cleared; 3) $v \in X$ and $v \in P_i$ with $|P_i| = 1$ and there are other vertices w in the bag with $w \in X$, then the automaton rejects the configuration, as v is the last vertex of P_i and may not be connected to any other partition anymore; 4) $v \in X$ is the last vertex of the bag that is contained in X, then the connected component is "done", the corresponding bits are cleared and one additional bit is set to indicate that the connected component cannot be extended anymore. When an edge $\{u, v\}$ is introduced, components might need to be merged. Assume $u, v \in X$, $u \in P_i$, and $v \in P_j$ with $i < j$ (otherwise, an edge-bag does not change the state), then we essentially perform a classical union-operation from the well-known union-find data structure. Hence, we assign all vertices that are assigned to P_j to P_i. Finally, at a join-bag we may join two states that agree locally on the vertices that are in X (i. e., they have assigned the same vertices to some P_i), however, they do not have to agree in the way the different vertices are assigned to

P_i (in fact, there does not have to be an isomorphism between these assignments). Therefore, the transition at a join-bag has to connect the corresponding components analogous to the edge-bags – in terms of Section 3.1 this transition is not symmetric. The description of the remaining quantifiers and subformulas is very similar.

4.4 Extending the Model-Checker to Optimization Problems

As the example formulas from the previous section already indicate, performing model-checking alone will not suffice to express many natural problems. In fact, every graph is a model of the formula $\tilde{\phi}_{\mathrm{vc}}$ if S simply contains all vertices. It is therefore a natural extension to consider an optimization version of the model-checking problem, which is usually formulated as follows [9, 13]: Given a logical structure $\mathcal{S}$, a formula $\phi(X_1,\dots,X_p)$ of the MSO-fragment defined in the previous section with free unary second-order variables $X_1,\dots,X_p$, and weight functions $\omega_1,\dots,\omega_p$ with $\omega_i\colon U\to\mathbb{Z}$; find $S_1,\dots,S_p$ with $S_i\subseteq U$ such that $\sum_{i=1}^{p}\sum_{s\in S_i}\omega_i(s)$ is *minimized* under $\mathcal{S}\models\phi(S_1,\dots,S_p)$, or conclude that $\mathcal{S}$ is not a model for ϕ for any assignment of the free variables. We can now correctly express the (actually *weighted*) optimization version of VERTEX-COVER as follows: $\phi_{\mathrm{vc}}(S)=\forall x\forall y\; E(x,y)\rightarrow\big(S(x)\vee S(y)\big)$. Similarly we can describe the optimization version of DOMINATING-SET if we assume the input does not have isolated vertices (or is reflexive), and we can also fix the formula $\tilde{\phi}_{\mathrm{fvs}}$:

$$\phi_{\mathrm{ds}}(S)=\forall x\exists y\; E(x,y)\wedge\big(S(x)\vee S(y)\big),\quad \phi_{\mathrm{fvs}}(S)=\exists^{\mathrm{forest}}F\;\forall x\;\big(S(x)\vee F(x)\big).$$

We can also *maximize* the term $\sum_{i=1}^{p}\sum_{s\in S_i}\omega_i(s)$ by multiplying all weights with -1 and, thus, express problems such as INDEPENDENT-SET: $\phi_{\mathrm{is}}(S)=\forall x\forall y\; E(x,y)\rightarrow\big(\neg S(x)\vee\neg S(y)\big)$. The implementation of such an optimization is straightforward: essentially there is a partition quantifier for every free variable X_i that partitions the universe into X_i and $\bar{X}_i$. We assign a current value of $\sum_{i=1}^{p}\sum_{s\in S_i}\omega_i(s)$ to every state of the automaton, which is adapted if elements are "added" to some of the free variables at introduce nodes. Note that, since we optimize an affine function, this does not increase the state space: even if multiple computational paths lead to the same state with different values at some node of the tree, it is well defined which of these values is the optimal one. Therefore, the cost of optimization only lies in the partition quantifier, i. e., we pay with k bits in the state description of the automaton per free variable – independently of the weights.

4.5 Handling Symmetric and Non-Symmetric Joins

In Section 4.3 we have defined the states of our automaton with respect to a formula, the left side of Table 1 gives an overview of the number of bits we require for the different parts of the formula. Let $\mathrm{bit}(\phi,k)$ be the number of bits that we have to reserve for a formula ϕ and a tree decomposition of maximum bag size k, i. e., the sum over the required bits of each part of the formula. By Observation 4 this implies that we can simulate the automaton (and hence, solve the model-checking problem) in time $O^*\big((2^{\mathrm{bit}(\phi,k)})^2\cdot n\big)$; or by Observation 6 in time $O^*\big(2^{\mathrm{bit}(\phi,k)}\cdot n\big)$ if the automaton is symmetric[2]. Unfortunately, this is not always the case, in fact, only the quantifier $\exists^{\mathrm{partition}}X_1,\dots,X_q$, the bits needed to optimize over free variables, as well as the formulas that do not require any bits, yield an symmetric tree automaton. That means that the simulation is wasteful if we consider a mixed formula (for instance, one that contains a partition and a connected quantifier). To overcome this issue, we partition

[2] The notation O^* supresses polynomial factors.

Table 1 The left table shows the precise number of bit we reserve in the description of a state of the tree automaton for different quantifier and formulas. The values are with respect to a tree decomposition with maximum bag size k. The right table gives an overview of example formulas ϕ used here, together with values symmetric(ϕ, k) and asymmetric(ϕ, k), as well as the precise time our algorithm will require for that particular formula.

Quantifier / Formula	Number of Bit
free variables $X_1, \ldots, X_q$	$q \cdot k$
$\exists^{\text{partition}} X_1, \ldots, X_q$	$k \cdot \log_2 q$
$\exists^{\text{connected}} X$	$k \cdot \log_2 k + 1$
$\exists^{\text{forest}} X$	$k \cdot \log_2 k$
$\forall x \forall y\ E(x,y) \to \chi_i$	0
$\forall x \exists y\ E(x,y) \wedge \chi_i$	k
$\exists x \forall y\ E(x,y) \to \chi_i$	$k+1$
$\exists x \exists y\ E(x,y) \wedge \chi_i$	1
$\forall x\ \chi_i$	0
$\exists x\ \chi_i$	1

ϕ	symmetric(ϕ, k) asymmetric(ϕ, k)	Time
ϕ_{3col}	$k \cdot \log_2(3)$ 0	$O^*(3^k)$
$\phi_{\text{vc}}(S)$	k 0	$O^*(2^k)$
$\phi_{\text{ds}}(S)$	k k	$O^*(8^k)$
$\phi_{\text{triangle-minor}}$	0 $3k \cdot \log_2(k) + 3$	$O^*(k^{6k+6})$
$\phi_{\text{fvs}}(S)$	k $k \cdot \log_2(k)$	$O^*(2^k k^{2k})$

the bits of the state description into two parts: first the "symmetric" bits of the quantifiers $\exists^{\text{partition}} X_1, \ldots, X_q$ and the bits required for optimization, and in the "asymmetric" ones of all other elements of the formula. Let symmetric(ϕ, k) and asymmetric(ϕ, k) be defined analogously to bit(ϕ, k). We implement the join of states as in the following lemma, allowing us to deduce the running time of the model-checker for concrete formulas. The right side of Table 1 provides an overview for formulas presented here.

▶ **Lemma 7.** *Let x be a node of T with children y and z, and let Q_y and Q_z be sets of states in which the automaton may be at y and z. Then the set Q_x of states in which the automaton may be at node x can be computed in time $O^*\big(2^{\text{symmetric}(\phi,k)+2\cdot\text{asymmetric}(\phi,k)}\big)$.*

Proof. To compute Q_x, we first split Q_y into $B_1, \ldots, B_q$ such that all elements in one B_i share the same "symmetric bits". This can be done in time $|Q_y|$ using bucket-sort. Note that we have $q \leq 2^{\text{symmetric}(\phi,k)}$ and $|B_i| \leq 2^{\text{asymmetric}(\phi,k)}$. With the same technique we identify for every elements v in Q_z its corresponding partition B_i. Finally, we compare v with the elements in B_i to identify those for which there is a transition in the automaton. This yields a running time of $|Q_z| \cdot \max_{i=1}^{q} |B_i| \leq 2^{\text{bit}(\phi,k)} \cdot 2^{\text{asymmetric}(\phi,k)} = 2^{\text{symmetric}(\phi,k)+2\cdot\text{asymmetric}(\phi,k)}$. ◀

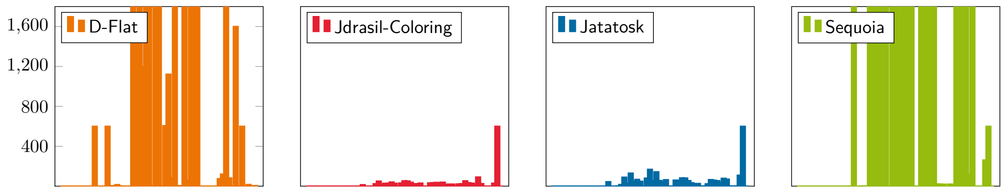
5 Applications and Experiments

To show the feasibility of our approach, we have performed experiments for widely investigated graph problems: 3-COLORING, VERTEX-COVER, DOMINATING-SET, INDEPENDENT-SET, and FEEDBACK-VERTEX-SET. All experiments were performed on an Intel Core processor containing four cores of 3.2 GHz each and 8 Gigabyte RAM. Jdrasil was used with Java 1.8 and both Sequoia and D-Flat were compiled with gcc 7.2. The implementation of Jatatosk uses hashing to realize Lemma 7, which works well in practice. We use a data set assembled from different sources containing graphs with 18 to 956 vertices and treewidth 3 to 13. The first source is a collection of transit graphs from GTFS-transit feeds [15] that was also used for experiments in [12], the second source are real-world instances collected in [2], and the last one are those of the PACE challenge [18] with treewidth at most 11. For 3-COLORING, the results are shown in Experiment 1.

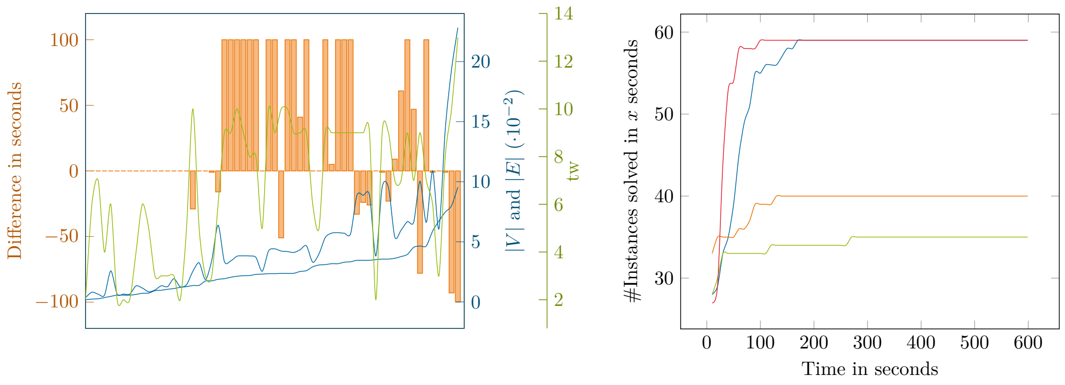
Experiment 1 3-COLORING.

	D-Flat	Jdrasil-Coloring	Jatatosk	Sequoia
Average Time	478.19	**36.52**	42.63	714.73
Standard Deviation	733.90	**77.8**	81.82	866.34
Median Time	**3.5**	21	24.5	20.5

(a) Average, standard deviation, and median of the time (in seconds) each solver needed to solve 3-COLORING over all instances of the data set. The best values are highlighted.



(b) Comparison of solvers for the 3-COLORING problem on the complete data set.



(c) The left picture shows the difference of Jatatosk against D-Flat and Sequoia. A positive bar means that Jatatosk is faster by this amount in seconds, and a negative bar means that either D-Flat or Sequoia is faster by that amount. The bars are capped at 100. On every instance, Jatatosk was compared against the solver that was faster on this particular instance. The image also shows for every instance the size and the treewidth of the input. The right image shows the number of instances that can be solved by each of the solvers in x seconds, i. e., faster growing functions are better. The colors in this image are as in (b).

6 Conclusion and Outlook

We investigated the practicability of dynamic programming on tree decompositions, which is arguably one of the corner stones of parameterized complexity theory. We implemented a simple interface for such programs and used it to build a competitive graph coloring solver with just a few lines of code. We hope that this interface allows others to implement and explore various dynamic programs. The whole power of these algorithms is well captured by Courcelle's Theorem, which states that there is an efficient version of such a program for every problem definable in monadic second-order logic. We took a step towards practice by implementing a "lightweight" version as model-checker for a small fragment of the logic. This fragment turns out to be powerful enough to express many natural problems.

References

1 Michael Abseher, Bernhard Bliem, Günther Charwat, Frederico Dusberger, Markus Hecher, and Stefan Woltran. D-flat: progress report. *DBAI, TU Wien, Tech. Rep. DBAI-TR-2014–86*, 2014.

2 Michael Abseher, Frederico Dusberger, Nysret Musliu, and Stefan Woltran. Improving the efficiency of dynamic programming on tree decompositions via machine learning. In *Proc. IJCAI*, pages 275–282, 2015.

3 M. Bannach. Jatatosk. `https://github.com/maxbannach/Jatatosk`, 2018. [Online; accessed 22-04-2018].

4 M. Bannach. Jdrasil for Graph Coloring. `https://github.com/maxbannach/Jdrasil-for-GraphColoring`, 2018. [Online; accessed 22-04-2018].

5 M. Bannach, S. Berndt, and T. Ehlers. Jdrasil. `https://github.com/maxbannach/Jdrasil`, 2017. [Online; accessed 22-04-2018].

6 Max Bannach, Sebastian Berndt, and Thorsten Ehlers. Jdrasil: A modular library for computing tree decompositions. In *16th International Symposium on Experimental Algorithms, SEA 2017, June 21-23, 2017, London, UK*, pages 28:1–28:21, 2017. `doi:10.4230/LIPIcs.SEA.2017.28`.

7 Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

8 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. `doi:10.1007/978-3-319-21275-3`.

10 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

11 M. R. Fellows. Parameterized complexity for practical computing. `http://www.mrfellows.net/wordpress/wp-content/uploads/2017/11/FellowsToppforsk2017.pdf`, 2018. [Online; accessed 22-04-2018].

12 Johannes Klaus Fichte, Neha Lodha, and Stefan Szeider. Sat-based local improvement for finding tree decompositions of small width. In *Theory and Applications of Satisfiability Testing - SAT*, pages 401–411, 2017.

13 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. Springer, 2006. `doi:10.1007/3-540-29953-X`.

14 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Annals of pure and applied logic*, 130(1-3):3–31, 2004.

15 gtfs2graphs - A Transit Feed to Graph Format Converter. `https://github.com/daajoe/gtfs2graphs`. Accessed: 2018-04-20.

16 Joachim Kneis, Alexander Langer, and Peter Rossmanith. Courcelle's theorem – a game-theoretic approach. *Discrete Optimization*, 8(4):568–594, 2011. `doi:10.1016/j.disopt.2011.06.001`.

17 Alexander Langer. *Fast algorithms for decomposable graphs*. PhD thesis, RWTH Aachen, 2013.

18 The Parameterized Algorithms and Computational Experiments Challenge (PACE). `https://pacechallenge.wordpress.com/`. Accessed: 2018-04-20.

19 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. In *Proc. ESA*, pages 68:1–68:13, 2017.

Average Whenever You Meet: Opportunistic Protocols for Community Detection

Luca Becchetti
Sapienza Università di Roma, Italy
becchetti@dis.uniroma1.it

Andrea Clementi
Università di Roma "Tor Vergata", Italy
clementi@mat.uniroma2.it

Pasin Manurangsi
U.C. Berkeley, California, USA
pasin@berkeley.edu

Emanuele Natale
Simons Institute and MPII, Germany
enatale@mpi-inf.mpg.de

Francesco Pasquale
Università di Roma "Tor Vergata", Italy
pasquale@mat.uniroma2.it

Prasad Raghavendra
U.C. Berkeley, California, USA
raghavendra@berkeley.edu

Luca Trevisan
Simons Institute and U.C. Berkeley, California, USA
luca@berkeley.edu

Abstract

Consider the following asynchronous, opportunistic communication model over a graph G: in each round, one edge is activated uniformly and independently at random and (only) its two endpoints can exchange messages and perform local computations. Under this model, we study the following random process: *The first time a vertex is an endpoint of an active edge, it chooses a random number, say ± 1 with probability $1/2$; then, in each round, the two endpoints of the currently active edge update their values to their average.*

We provide a rigorous analysis of the above process showing that, if G exhibits a two-community structure (for example, two expanders connected by a sparse cut), the values held by the nodes will collectively reflect the underlying community structure over a suitable phase of the above process. Our analysis requires new concentration bounds on the product of certain random matrices that are technically challenging and possibly of independent interest.

We then exploit our analysis to design the first opportunistic protocols that approximately recover community structure using only logarithmic (or polylogarithmic, depending on the sparsity of the cut) work per node.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Community Detection, Random Processes, Spectral Analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.7

© Luca Becchetti, Andrea Clementi, Pasin Manurangsi, Emanuele Natale, Francesco Pasquale, Prasad Raghavendra, and Luca Trevisan;
licensed under Creative Commons License CC-BY

Related Version A full version of the paper is available at [2], `https://arxiv.org/abs/1703.05045`.

Funding This material is based upon work supported by the National Science Foundation under Grants No. 1540685 and No. 1655215 and by the University of "Tor Vergata" under research programme "Mission: Sustainability" project ISIDE (grant no. E81I18000110005).

1 Introduction

The Averaging Protocol. Consider the following, elementary distributed process on an undirected graph $G = (V, E)$ with $|V| = n$ nodes and $|E| = m$ edges. Each node v holds a real number x_v (which we call the *state* of node v); at each time step, one random edge $\{u, v\}$ becomes active and its endpoints u and v update their states to their average.

Viewed as a protocol, the above process is consistent with asynchronous, opportunistic communication models, such as those considered in [1] for *population protocols*; here, in every round, one edge is activated uniformly and independently at random and (only) its two endpoints can exchange messages and perform local computations in that round. We further assume no global clock is available (nodes can at most count the number of local activations) and that the network is *anonymous*, i.e., nodes are not aware of theirs or their neighbors' identities and all nodes run the same process at all times.

The long-term behavior of the process outlined above is well-understood: assuming G to be connected, for each initial global state $\mathbf{x} \in \mathbb{R}^V$ the system converges to a global state in which all nodes share a common value, namely, the average of their initial states. A variant of an argument of Boyd et al. [4] shows that convergence time is equivalent to the the mixing time of a lazy random walk on the graph, namely $\mathcal{O}\left(\frac{1}{\lambda_2} n \log n\right)$, where λ_2 is the second smallest eigenvalue of the normalized Laplacian of G.

Distributed Community Detection. Suppose now that G is *well-clustered*, i.e. it exhibits a *community structure* which in the simplest case consists of two equal-sized expanders, connected by a sparse cut: this structure arises, for instance, when the graph is sampled from the popular *stochastic block model* $\mathcal{G}_{n,p,q}$ for $p \gg q$ and $p \geqslant \log n/n$ [6, 7, 10]. If we let the averaging process unfold on such a graph, for example starting from an initial ± 1 random global state, one might reasonably expect a faster, transient convergence toward some local average within each community, accompanied by a slower, global convergence toward the average taken over the entire graph. If, as is likely the case, a gap exists between the local averages of the two communities, the global state during the transient phase would reflect the graph's underlying community structure. This intuition suggests the main questions we address in this paper: Is there *a phase in which the global state carries information about community structure*? If so, *how strong is the corresponding "signal"*? Finally, *can nodes leverage local history to recover this information*?

Our Results: Highlights. We show that, if G exhibits a two-community structure (for example, two expanders connected by a sparse cut), the values held by the nodes will collectively reflect the underlying community structure over a suitable phase of the above process, allowing efficient and effective recovery in important cases.

In more detail, we first provide a first moment analysis showing that, for a large class of almost-regular clustered graphs that includes the *stochastic block model*, the expected values held by all but a negligible fraction of the nodes eventually reflect the underlying cut signal. We prove this property emerges after a "mixing" period of length $\mathcal{O}(n \log n)$.

We further provide a second moment analysis for a more restricted class of regular clustered graphs that includes the *regular stochastic block model* [3, 5, 11]. Since nodes do not share a common clock, it is not immediate to translate the above results into distributed clustering protocols. To this purpose, we show that concentration holds over a long time window and most nodes are able to select a local time within this window. So, most nodes can efficiently and locally identify their community of reference over a suitable time window. Even for the above class of regular graphs, our second moment analysis requires new concentration bounds on the product of certain random matrices that are technically challenging and possibly of independent interest.

This results in the first opportunistic protocols that approximately recover community structure. For clustered graphs with sparse (i.e. size $o(m)$) cut, we devise a first protocol, using the sign of the nodes' state as local clustering criterion (see Algorithm 2), that converges in $\mathcal{O}(n \log n)$ time and has only polylogarithmic work per node (see Theorem 12 for a formal statement). So, the protocol can be much faster than the global mixing time of the corresponding process and, moreover, the work per node does not depend on the node degree, thus resulting very efficient in the case of dense graphs. For clustered graphs with dense cut (i.e. size $\Theta(m)$), the cut "signal" is much harder to recover: we derive a more complex second moment analysis leading us to a weighted version of the averaging process, equipped with a clustering criterion based on the fluctuations of the nodes' state. This second protocol (see Algorithm 3) converges within $\mathcal{O}(n \log n + n/\lambda_2)$ rounds and has work per node $\mathcal{O}(\operatorname{polylog}(n) + 1/\lambda_2)$ (see Theorem 14, Corollaries 15 and 16 for formal statements).

Comparison to Previous Work. We here discuss only strongly-related work (see the full-version [2] for a more detailed description of previous results. The idea of using averaging local rules to perform distributed community detection is not new: In [3], Becchetti et al. consider a deterministic dynamics in which, at every round, each node updates its local state to the average of its neighbors. The authors show that this results in a fast clustering algorithm with provable accuracy on a wide class of almost-regular graphs that includes the stochastic block model. We remark that the algorithm in [3] (only) works in a *synchronous, parallel communication model* where every node exchanges data with all its neighbors in each round. This implies considerable work and communication costs, especially when the graph is dense. It turns out that, in d-regular, well-clustered graphs, the algorithm in [3] requires overall communication cost $\Theta(nd \operatorname{polylog}(n))$ and work per-node $\Theta(d \operatorname{polylog}(n))$. On the other hand, each step of the process in [3] is described by the same matrix and its deterministic evolution unfolds according to the power of this matrix applied to the initial state. In contrast, the averaging process we consider in this paper is considerably harder to analyze than the one in [3], since each step is described by a random, possibly different averaging matrix. Differently from [3], our goal here is the design of simple, lightweight protocols for fully-decentralized community detection which fit the asynchronous, opportunistic communication model, in which a (random) link activation represents an opportunistic meeting that the endpoints can exploit to exchange one-to-one messages. More specifically, by "lightweight" we mean protocols that require minimalistic assumptions as to network capabilities, while performing their task with minimal work, storage and communication per node (at most logarithmic or polylogarithmic in our case). In this respect, any clustering strategies (like the one in [12]) which construct (and then work over) some static, sparse subgraph of the underlying graph are unfeasible in the opportunistic model we consider here. This restrictive setting is motivated by network scenarios in which individual agents need to autonomously and locally uncover underlying, implicit communities

Algorithm 1: Updating rule for a node u of an active edge, where $\delta \in (0, 1)$ is the parameter measuring the weight given to the neighbor's value.

```
AVERAGING($\delta$)   (for a node $u$ that is one of the two endpoints of an active edge)
Initialization: If it is the first time $u$ is active, then pick $\mathbf{x}_u \in \{-1, +1\}$ u.a.r.
Update: Send $\mathbf{x}_u$ to the other endpoint and set $\mathbf{x}_u := (1 - \delta)\mathbf{x}_u + \delta r$, where $r$ is the
 value received from the other endpoint.
```

of which they are members. This has widespread applicability, for example in communication systems where lightweight data can be locally shared via wireless opportunistic meetings when agents come within close range [13].

Roadmap of the paper. After presenting some preliminaries in Section 2, the first moment analysis for almost-regular graphs is given in Section 3. The second moment analysis for regular graphs is given in Section 4 while, in Section 5, we describe our protocols for community detection and give the main bounds on their performances. Due to space constraints, most technical results are given in the full-version of the paper [2].

2 Preliminaries

Averaging process. In general, we consider the weighted version of the averaging process described in the introduction: In each round, one edge of the graph is sampled uniformly at random and the two endpoints of the sampled edge execute Algorithm 1.

Graphs and their spectra. For a connected graph $G = (V, E)$ with n nodes, m edges and adjacency matrix A, let $0 = \lambda_1 \leqslant \cdots \leqslant \lambda_n$ be the eigenvalues of the normalized Laplacian $\mathcal{L} = I - D^{-1/2}AD^{-1/2}$, where D is the diagonal matrix with the degrees of the nodes. We consider the following graph classes.

▶ **Definition 1.** An (n, d, β)*-almost-regular graph* is a connected, non-bipartite graph $G = (V, E)$ with n nodes such that every node has degree $d \pm \beta d$. An (n, d, b)*-clustered regular graph*, where n is even and d and b are two positive integers with $2b < d < n$, is a graph $G = ((V_1, V_2), E)$ over node set $V = V_1 \cup V_2$, with $|V_1| = |V_2| = n/2$ and such that: (i) every node has degree d and (ii) every node in V_1 has b neighbors in V_2 and every node in V_2 has b neighbors in V_1.

It is easy to see that the indicator vector $\boldsymbol{\chi} \in \{-1, +1\}$ of the cut (V_1, V_2) is an eigenvector of $\mathcal{L}$ with eigenvalue $\frac{2b}{d}$, whenever the graph is clustered regular. If we further assume that $\lambda_3 > \frac{2b}{d}$, then $\boldsymbol{\chi}$ is an eigenvector of λ_2.

Block reconstruction. We next discuss what it means to recover the "underlying community structure" in a distributed setting, a notion that can come in stronger or weaker flavors [6, 11, 8, 9]. Ideally, we would like the protocol to reach a state in which, at least with high probability, each node can use a simple rule to assign itself one of two possible labels, so that labelling within each community is consistent and nodes in different communities are assigned different labels. Achieving this corresponds to *exact (block) reconstruction*. The next best guarantee is *weak (block) reconstruction*.

▶ **Definition 2** (Weak Reconstruction)**.** A function $f : V \to \{\pm 1\}$ is said to be an *ε-weak reconstruction* of G if subsets $W_1 \subseteq V_1$ and $W_2 \subseteq V_2$ exist, each of size at least $(1-\varepsilon)n/2$, such that $f(W_1) \cap f(W_2) = \emptyset$.

We introduce a third notion, which we call *community-sensitive labeling* (CSL for short): in this case, there is a predicate that can be applied to pairs of labels so that, for all but a small fraction of outliers, the labels of any two nodes within the same community satisfy the predicate, whereas the converse occurs when they belong to different communities[1]. In this paper, informally speaking, nodes are labelled with binary signatures of logarithmic length, while two labels satisfy the predicate whenever their Hamming distance is below a certain threshold. This introduces a notion of similarity between nodes of the graph, with labels behaving like profiles that reflect community membership[2]. Note that this weaker notion of community-detection allows nodes to locally tell "friends" in their community from "foes" in the other community, which is the main application of distributed community detection in the opportunistic setting we consider here.

Let $\Delta(\mathbf{x}, \mathbf{y})$ denote the *Hamming distance* between two binary strings $\mathbf{x}$ and $\mathbf{y}$.

▶ **Definition 3** (Community-sensitive labeling)**.** Let $G = (V, E)$ be a graph, let (V_1, V_2) be a partition of V and let $\gamma \in (0, 1]$. For some $\ell \in \mathbb{N}$, a function $\mathbf{h} : V_1 \cup V_2 \to \{0, 1\}^\ell$ is a *γ-community-sensitive labeling* for (V_1, V_2) if a subset $\tilde{V} \subseteq V$ with size $|\tilde{V}| \geqslant (1-\gamma)|V|$ and two constants $0 \leqslant c_1 < c_2 \leqslant 1$ exist, such that for all $u, v \in \tilde{V}$ it holds that: $\Delta(\mathbf{h}_u, \mathbf{h}_v) \leqslant c_1 \ell$ if $i_u = i_v$, and $\Delta(\mathbf{h}_u, \mathbf{h}_v) \geqslant c_2 \ell$, otherwise, where $i_u = 1$ if $u \in V_1$ and $i_u = 2$ if $u \in V_2$.

3 First moment analysis

We analyze the expected behaviour of Algorithm Averaging(1/2) on an almost-regular graph G. The evolution of the resulting process can be formally described by the recursion $\mathbf{x}^{(t+1)} = W_t \cdot \mathbf{x}^{(t)}$, where $W_t = (W_t(i,j))$ is the random matrix that defines the updates of the values at round t, i.e.,

$$W_t(i,j) = \begin{cases} 0 & \text{if } i \neq j \text{ and } \{i,j\} \text{ is not sampled (at round } t), \\ 1/2 & \begin{array}{l}\text{if } i = j \text{ and an edge with endpoint } i \text{ is sampled} \\ \text{or } i \neq j \text{ and edge } \{i,j\} \text{ is sampled,}\end{array} \\ 1 & \text{if } i = j \text{ and } i \text{ is not an endpoint of sampled edge.} \end{cases} \tag{1}$$

and the initial random vector $\mathbf{x}^{(0)}$ is uniformly distributed in $\{-1,1\}^n$.[3] Note that, consequently, $\mathbf{x}^{(t+1)} = W_t \cdots W_1 \mathbf{x}^{(0)}$, with the W_i's independently and identically distributed. Simple calculus shows that the expectation of the random matrices $\{W_t : t \geqslant 0\}$ can be expressed as

$$\overline{W} := \mathbb{E}[W_t] = I - \frac{1}{2m} L \,, \tag{2}$$

where $L = D - A$ is the Laplacian matrix of G. Matrix $\overline{W}$ is thus symmetric and doubly-stochastic. We denote its eigenvalues as $\bar{\lambda}_1, \ldots, \bar{\lambda}_n$, with $1 = \bar{\lambda}_1 \geqslant \bar{\lambda}_2 \geqslant \cdots \bar{\lambda}_n \geqslant -1$.

[1] Note that a weak reconstruction protocol entails a community-sensitive labeling. In this case, the predicate is true if two labels are the same.

[2] Hence the phrase *community-sensitive Labeling* we use to refer to our approach.

[3] Notice that, since each node chooses value ± 1 with probability $1/2$ the first time it is active, by using the principle of deferred decisions we can assume there exists an "initial" random vector $\mathbf{x}^{(0)}$ uniformly distributed in $\{-1,+1\}^n$.

Our first contribution is an analysis of the expected evolution of the averaging process over (n, d, β)-almost regular graphs that possess a hidden and balanced partition of the nodes with the following properties: (i) The cut separating the two communities contains $o(m)$ edges; (ii) the subgraphs induced by the two communities are expanders, i.e., the gap $\lambda_3 - \lambda_2$ is constant. The above conditions on the underlying graph are satisfied, for instance, by graphs sampled from the stochastic block model[4] $\mathcal{G}_{n,p,q}$ for $q = o(p)$ and $p \geqslant \log n/n$. Our analysis proves the following results.

▶ **Theorem 4.** *Let $G = (V, E)$ be an (n, d, β)-almost regular graph $G = (V, E)$ with a balanced partition $V = (V_1, V_2)$ and such that: (i) The cut $E(V_1, V_2)$ is sparse, i.e., $m_{1,2} = |E(V_1, V_2)| = o(m)$; (ii) $\lambda_3 - \lambda_2 = \Omega(1)$. If nodes of G execute Protocol* Averaging *then, with constant probability w.r.t. the initial random vector $\mathbf{x}^{(0)} \in \{-1, 1\}^n$, after $\Theta(n \log n)$ rounds the following holds for all but $o(n)$ nodes:*
(i) The expected value of a node u increases or decreases depending on the community it belongs to, i.e., $\mathbf{sgn}\left(\mathbb{E}\left[\mathbf{x}_u^{(t-1)} \,|\, \mathbf{x}^{(0)}\right] - \mathbb{E}\left[\mathbf{x}_u^{(t)} \,|\, \mathbf{x}^{(0)}\right]\right) = \mathbf{sgn}\left(\boldsymbol{\chi}_u\right)$;
(ii) Over a suitable time window of length $\Omega(n \log n)$, the sign of the expected value of a node u reflects the community u belongs to, i.e., $\mathbf{sgn}\left(\mathbb{E}\left[\mathbf{x}_u^{(t)} \,|\, \mathbf{x}^{(0)}\right]\right) = \mathbf{sgn}\left(\alpha_2 \boldsymbol{\chi}_u\right)$, for some $\alpha_2 = \alpha_2(\mathbf{x}^{(0)})$.

We note that these results suggest two different local criteria for community-sensitive labeling: (i) According to the first one, every node uses the sign of its own state within the aforementioned time window to set the generic component of its binary label (in fact, we run independent copies of the averaging process to get binary labels of logarithmic size - see Protocol Sign-Labeling in Section 5.1). (ii) According to the second criterion, every node uses the signs of fluctuations of its own value along consecutive rounds to set the generic component of its binary label (see Protocol Jump-Labeling in Section 5.2).

The above results describe the "expected" behaviour of the averaging process over a large class of well-clustered graphs, at the same time showing that our approach might lead to efficient, opportunistic protocols for block reconstruction. Yet, designing and analyzing protocols with provable, probabilistic guarantees, requires addressing the following questions: i) Do realizations of the averaging process approximately follow its expected behavior with high, or even constant, probability? ii) If this is the case, how can nodes locally and asynchronously recover the cut signal, let alone guess the "right" global time window? The first issue is addressed in Section 4, while the second one is addressed in Section 5, which presents our main algorithmic results for community detection.

4 Second Moment Analysis

Recall from Section 3 that $\mathbf{x}^{(t)}$ depends on the product of t identically distributed random matrices. Not much is known about concentration of such products, but we are able to accurately characterize the class of regular clustered graphs. We point out that many of the technical results and tools we develop to this purpose apply to far more general settings than the regular case and may be of independent interest. In more detail, we are able to provide accurate concentration bounds on the norm of $\mathbf{x}^{(t)}$'s projection onto the subspace spanned by the first and second eigenvector of $\overline{W}$ for a class of regular clustered graphs that includes the *regular stochastic block model* [3, 5, 11]. These bounds are derived separately for two different

[4] See the full-version [2] for the definition of $\mathcal{G}_{n,p,q}$ and for more details about our results for $\mathcal{G}_{n,p,q}$.

regimes, defined by the sparseness of the cut separating the two communities. Assuming good inner expansion within each community, the first concentration result applies for cuts of size $o(m/\log^2 n)$ and it is given in Subsectioni 4.1 while, for the case of cuts of size up to αm for any $\alpha < 1$, the obtained concentration results are described in Subsection 4.2.

4.1 Second moment analysis for sparse cuts

We next provide a second moment analysis of the AVERAGING(δ) with $\delta = 1/2$ on the class of (n, d, b)-*clustered regular graphs* when the cut between the two communities is relatively sparse, i.e., for $\lambda_2 = 2b/d = o(\lambda_3/\log n)$. This analysis is consistent with the "expected" clustering behaviour of the dynamics explored in the previous section and highlights clustering properties that emerge well before global mixing time, as we show in Section 5.1.

Restriction to (n, d, b)-*clustered regular graphs* simplifies the analysis of the AVERAGING dynamics. When G is regular, $\overline{W}$ defined in (2) can be written as $\overline{W} = \left(1 - \frac{1}{n}\right) I + \frac{1}{n} P = I - \frac{1}{n}\mathcal{L}$. This obviously implies that $\overline{W}$ and $\mathcal{L}$ share the same eigenvectors, while every eigenvalue λ_i of $\mathcal{L}$ corresponds to an eigenvalue $\bar{\lambda}_i = 1 - \lambda_i/n$ of $\overline{W}$. For (n, d, b)-*clustered regular graphs*, these facts further imply $\bar{\lambda}_2 = 1 - \lambda_2/n = 1 - 2b/dn$ whenever $\lambda_3 > \frac{2b}{d}$ while, very importantly, the partition indicator vector $\boldsymbol{\chi}$ turns out to be the eigenvector of $\overline{W}$ corresponding to $\bar{\lambda}_2$ (see (2)). On the other hand, even in this restricted setting, our second moment analysis requires new, non-standard concentration results for the product of random matrices that apply to far more general settings and may be of independent interest.

For the sake of readability, in the remainder we denote $\mathbf{x}^{(t)}$'s projection onto $\mathbf{1}$ by $\mathbf{x}_{\|}$ and we use $\mathbf{y}^{(t)}$ to denote its component in the eigenspace of the second eigenvalue of $\overline{W}$ (i.e., $\boldsymbol{\chi}$).[5] Finally, we use $\mathbf{z}^{(t)}$ to denote $\mathbf{x}^{(t)}$'s projection onto the subspace orthogonal to $\mathbf{1}$ and $\boldsymbol{\chi}$. We thus have:

$$\mathbf{x}^{(t)} = \mathbf{x}_{\|} + \mathbf{y}^{(t)} + \mathbf{z}^{(t)}. \tag{3}$$

Our analysis of the process induced by AVERAGING(1/2) provides the following bound, whose proof can be found in the full-version [2].

▶ **Theorem 5** (Second moment analysis)**.** *Let G be an (n, d, b)-clustered regular graph with $\lambda_2 = \frac{2b}{d} = o\left(\lambda_3/\log n\right)$. Then, for every $\frac{3n}{\lambda_3}\log n \leqslant t \leqslant \frac{n}{4\lambda_2}$ it holds that*

$$\mathbb{E}\left[\left\|\mathbf{y}^{(t)} + \mathbf{z}^{(t)} - \mathbf{y}^{(0)}\right\|^2\right] \leqslant \frac{3\lambda_2 t}{n}.$$

We prove Theorem 5 by bounding and tracking the lengths of the projections of $\mathbf{x}^{(t)}$ onto the eigenspace of λ_2 and onto the space orthogonal to $\mathbf{1}$ and $\boldsymbol{\chi}$, i.e. $\|\mathbf{y}^{(t)}\|^2$ and $\|\mathbf{z}^{(t)}\|^2$. We here remark that the only part using the regularity of the graph is the derivation of the upper bound on $\mathbb{E}\left[\|\mathbf{y}^{(t+1)}\|^2\right]$, in particular its second addend. This term arises from an expression involving the Laplacian of G, which is far from simple in general, but that very nicely simplifies in the regular case. We suspect that increasingly weaker bounds should be achievable as the graph deviates from regularity.

Theorem 5 gives an upper bound on the squared norm of the difference of the state vector at step t with the state vector at step 0. Intuitively, this will allow us to conclude that, for most vertices, $\mathbf{x}_v^{(t)} \approx \mathbf{x}_{\|,v} + \mathbf{y}_v^{(0)}$ over a time window of size $\Omega(n \log n)$. More formally, Corollary 7 below shows how such a *global* bound can be used to derive *pointwise* bounds on the values of the nodes.

[5] Note that $\mathbf{x}_{\|}$ is time-invariant.

▶ **Definition 6.** A node v is *ε-good* at time t if $(\mathbf{x}_v^{(t)} - (\mathbf{x}_{\|,v} + \mathbf{y}_v^{(0)}))^2 \leqslant \frac{\varepsilon^2}{n}\|\mathbf{y}^{(0)}\|^2$, it is *$\varepsilon$-bad* otherwise. We also define $B_t = \{u : u \text{ is } \varepsilon\text{-bad at time } t\}$.

▶ **Corollary 7.** *Assume $3\frac{n}{\lambda_3}\log n \leqslant t \leqslant 3c\frac{n}{\lambda_3}\log n$ for any absolute constant $c \geqslant 1$ and $\lambda_2/\lambda_3 \leqslant \varepsilon^4/(4c\log n)$:*

$$\mathbb{P}\left[|B_t| > \varepsilon n \,|\, \mathbf{x}^{(0)} = \mathbf{x}\right] \leqslant \varepsilon. \tag{4}$$

The next lemma strengthens the result above, giving a bound on the number of nodes that are good over a relatively large time-window. This is the key-property that we leverage to analyse the asynchronous protocol Sign-Labeling. The main idea of its proof is to first show that with probability strictly larger than $1-\varepsilon$, the number of ε-good nodes is at least $n\cdot(1-\varepsilon/\log n)$ in every round $t \in [t_1, 2t_1]$. Theorem 5 already ensures this to be true in any given time step within a suitable window, but simply taking a union bound will not work, since we have $n\log n$ time steps and only a $1-\varepsilon$ probability of observing the desired outcome in each of them. We will instead argue about the possible magnitude of the change in $\|\mathbf{y}^{(t)} + \mathbf{z}^{(t)} - \mathbf{y}^{(0)}\|^2$ over time, assuming this quantity is small at time $6\frac{n}{\lambda_3}\log n$. We will then show that our argument implies that, with probability $1-\varepsilon$, at least $n - \varepsilon n$ nodes remain ε-good over the entire window $[6\frac{n}{\lambda_3}\log n, 12\frac{n}{\lambda_3}\log n]$.

▶ **Lemma 8** (Non-ephemeral good nodes). *Let $\varepsilon > 0$ be an arbitrarily small value, let G be an (n,d,b)-clustered regular graph with $\frac{\lambda_2}{\lambda_3} \leqslant \frac{\lambda_3\varepsilon^4}{c\log^2 n}$, for a large enough costant c. If we execute* Averaging$(1/2)$ *on G, it holds that*
$\mathbb{P}\left[|B_t| \leqslant 3\varepsilon\cdot n\,,\, \forall t\,:\, 6\frac{n}{\lambda_3}\log n \leqslant t \leqslant 12\frac{n}{\lambda_3}\log n\right] \geqslant 1-\varepsilon.$

4.2 Second moment analysis for dense cuts

In this section, we extend our study to the lazy averaging algorithm Averaging(δ) where $\delta < 1/2$. Similar to the previous section, we assume that the underlying graph G is an (n,d,b)-clustered regular graph and $\lambda_3 > \lambda_2 = 2b/d$. However, this new analysis works even for large (constant) λ_2, in contrast to that in Section 4.1 which only works for small $\lambda_2 \ll 1/\log^2 n$. Informally speaking, we show that, for an appropriate value of δ and any t such that $\Omega(n\log n) \leqslant t \leqslant \mathcal{O}(n^2)$, with large probability, the vector $\mathbf{y}^{(t)} + \mathbf{z}^{(t)}$ is almost parallel to χ, i.e., $\|\mathbf{z}^{(t)}\|$ is much smaller than $\|\mathbf{y}^{(t)}\|$. A more precise statement is given below as Theorem 9. Note that, for brevity, we write $\mathcal{E}$ here to denote the sequence $\{(u_t, v_t)\}_{t\in\mathbb{N}}$ of the edges chosen by the protocol.

▶ **Theorem 9.** *For any sufficiently large $n \in \mathbb{N}$, any[6] $\delta \in (0, 0.8(\lambda_3 - \lambda_2))$ and any $t \in \left[\Omega\left(\frac{n}{\delta(\lambda_3-\lambda_2)}\log(n/\delta)\right), \mathcal{O}\left(\frac{n^2}{\delta(\lambda_3-\lambda_2)}\left(\frac{d(\lambda_3-\lambda_2)}{\delta b}\right)^{2/3}\right)\right]$, we have*
$\Pr_{\mathbf{x}^{(0)},\mathcal{E}}\left[\|\mathbf{z}^{(t)}\|^2 \leqslant \sqrt{\frac{\delta b}{d(\lambda_3-\lambda_2)}}\|\mathbf{y}^{(t)}\|^2\right] \geqslant 1 - \mathcal{O}\left(\sqrt[3]{\frac{\delta b}{d(\lambda_3-\lambda_2)}} + \frac{1}{\sqrt{n}}\right).$

Theorem 9 should be compared to Theorem 5: both assert that $\|\mathbf{y}^{(t)}\|$ is much larger than $\|\mathbf{z}^{(t)}\|$, but Theorem 9 works even when λ_2 is quite large whereas Theorem 5 only holds for $\lambda_2 \ll 1/\log^2 n$. While the parameter dependencies in Theorem 9 may look confusing at first, there are mainly two cases that are interesting here. First, for any error parameter ε, we can pick δ depending only on ε and $\lambda_3 - \lambda_2$ in such a way that Theorem 9 implies that, with probability $1-\varepsilon$, $\|\mathbf{z}^{(t)}\|^2$ is at most $\varepsilon\|\mathbf{y}^{(t)}\|^2$, as stated below.

[6] Here 0.8 is arbitrary and can be changed to any constant less than 1. However, we pick an absolute constant here to avoid introducing another parameter to our theorem.

Algorithm 2: Sign-Labeling algorithm.

```
Sign-Labeling(T,$\ell$) (for a node $u$ that is one of the two endpoints of an active edge)
**Component selection:** Jointly with the other endpoint choose a component $j \in [\ell]$
  u.a.r.
**Initialization and update:** Run one step of Averaging (1/2) for component $j$.
**Labeling:** If this is the $T$-th activation of component $j$: set $\mathbf{h}_u^{sign}(j) = \mathbf{sgn}(\mathbf{x}_u(j))$.
```

▶ **Corollary 10.** *For any constant $\varepsilon > 0$ and for any $\lambda_3 > \lambda_2$, there exists δ depending only on ε and $\lambda_3 - \lambda_2$ such that, for any sufficiently large n and for any $t \in [\Omega_{\varepsilon,\lambda_3-\lambda_2}(n \log n), \mathcal{O}(n^2)]$, we have* $\Pr_{\mathbf{x}^{(0)},\mathcal{E}}\left[\|\mathbf{z}^{(t)}\|^2 \leqslant \varepsilon\|\mathbf{y}^{(t)}\|^2\right] \geqslant 1 - \varepsilon$.

Another interesting case is when $\delta = 1/2$ (i.e., we consider the basic averaging protocol). Recalling that $\lambda_2 = 2b/d$, observe that λ_2 appears in both the bound on $\|\mathbf{z}^{(t)}\|^2$ and the error probability. Hence, we can derive a similar lemma as the one above, but with λ_2 depending on ε instead of δ:

▶ **Corollary 11.** *Fix $\delta = 1/2$. For any constant $\varepsilon > 0$, any*[7] *$\lambda_3 > 0.7$, any sufficiently small λ_2 depending only on ε, any sufficiently large n and any $t \in [\Omega_\varepsilon(n \log n), \mathcal{O}(n^2)]$, we have* $\Pr_{\mathbf{x}^{(0)},\mathcal{E}}\left[\|\mathbf{z}^{(t)}\|^2 \leqslant \varepsilon\|\mathbf{y}^{(t)}\|^2\right] \geqslant 1 - \varepsilon$.

5 Distributed Community Detection

5.1 The Sign-Labeling protocol for sparse cuts

In the case of sparse cuts (i.e. of size $o(m/\log^2 n)$), the obtained bound on the variance of non-ephemeral nodes (see Lemma 8) holds over a time window that essentially equals the one “suggested” by our first moment analysis. Hence, we next propose a simple, lightweight opportunistic protocol that provides community-sensitive labeling for graphs that exhibit a relatively sparse cut.

The algorithm, denoted as Sign-Labeling (see Algorithm 2), adds a simple *labeling rule* to the Averaging(1/2) process: Each node keeps track of the number of times it is activated. Upon its T-th activation, for a suitable $T = \Theta(\log n)$, the node uses the sign of its current value as a binary label. The above local strategy is applied to ℓ independent runs of Averaging(1/2), so that every node is eventually assigned a binary signature of length ℓ.

Roughly, Lemma 8 implies that over a suitable time window of size $\Theta(n \log n)$, for all nodes u but a fraction $\mathcal{O}(\varepsilon/\log n)$, we have $\mathbf{sgn}(\mathbf{x}_u^{(t)}) = \mathbf{sgn}(\mathbf{x}_{\|,u} + \mathbf{y}_u^{(0)})$. Recalling that $\mathbf{x}_\|$ and $\mathbf{y}^{(0)}$ respectively are $\mathbf{x}^{(0)}$'s projections along $\boldsymbol{\chi}/\sqrt{n}$ and $\mathbf{1}/\sqrt{n}$, this immediately implies that, with probability $1 - \varepsilon$ and up to a fraction ε of the nodes, $\mathbf{sgn}(\mathbf{x}_u^{(t)}) = \mathbf{sgn}(\mathbf{x}_v^{(t)})$, whenever u and v belong to the same community and t falls within the aforementioned window. As to the latter condition, we prove that each node labels itself within the right window with probability at least $1 - 1/n$.[8] Moreover, $\mathbf{sgn}(\mathbf{x}_{\|,u} + \mathbf{y}_u^{(0)}))) = \mathbf{sgn}(\boldsymbol{\chi}_u)$, whenever $\mathbf{y}_u^{(0)}$ exceeds $\mathbf{x}_{\|,u}$ in modulus, which occurs with probability $1/2 - o(1)$ from the (independent) Rademacher initialization. As a consequence, if we run ℓ suitably independent copies of the process, the following will happen for all but a fraction $\mathcal{O}(\varepsilon)$ of the nodes: the signatures of

[7] 0.7 here can be replaced by any constant larger than 0.5.

[8] It may be worth noting that $\mathbf{sgn}(\mathbf{x}_u^{(t)}) = \mathbf{sgn}(\mathbf{x}_v^{(t)})$ for u and v belonging to the same community does not imply $\mathbf{sgn}(\mathbf{x}_u^{(t)}) \neq \mathbf{sgn}(\mathbf{x}_v^{(t)})$ when they don't.

Algorithm 3: Jump-Labeling Here, τ_u is a local counter keeping track of the number of times u was an endpoint of an active edge, while x_u is u's current value.

```
Jump-Labeling($\delta, \tau^{s}, \tilde{\tau}^{s}, \tau^{e}, \tilde{\tau}^{e}$)
(for a node $u$ that is one of the two endpoints of an active edge)
Initialization: The first time it is activated, $u$ chooses $\tau_u^{s}, \tau_u^{e} \in \mathbb{N}$ independently
  uniformly at random from $[\tau^{s}, \tilde{\tau}^{s}]$ and $[\tau^{e}, \tilde{\tau}^{e}]$ respectively. Moreover, let $\tau_u = 0$.
Update (and Averaging's initialization): Run one step of Averaging($\delta$).
Labeling: If $\tau_u = \tau_u^{s}$, then set $x_u^{s} = x_u$. If $\tau_u = \tau_u^{e}$, then label $\mathbf{h}_u^{jump} = \mathbf{sgn}(x_u^{s} - x_u)$.
```

two nodes belonging to the same community will agree on $\ell - o(1)$ bits, whereas those of two nodes belonging to different communities will disagree on $\Omega(\ell)$ bits, i.e., our algorithm returns a community-sensitive labeling of the graph, as stated in the following theorem and corollary.

▶ **Theorem 12** (Community-sensitive labeling). *Let $\varepsilon > 0$ be an arbitrarily small value, let G be an (n, d, b)-clustered regular graph with $\frac{\lambda_2}{\lambda_3} \leqslant \frac{\lambda_3 \varepsilon^4}{c \log^2 n}$, for a large enough constant c. Then, protocol* Sign-Labeling *(T, ℓ) with $T = (8/\lambda_3)\log n$ and $\ell = 10\varepsilon^{-1}\log n$ performs a γ-community-sensitive labeling of G according to Definition 3 with $c_1 = 4\varepsilon$, $c_2 = 1/6$ and $\gamma = 6\varepsilon$, w.h.p. The convergence time is $\mathcal{O}(n\ell \log n/\lambda_3)$ and the work per node is $\mathcal{O}(\ell \log n/\lambda_3)$, w.h.p.*

Notice that, according to the hypothesis of Theorem 12, in order to set local parameters T and ℓ, nodes should know parameters ε and λ_3 (in addition to a polynomial upper bound on the number of the nodes). However, it easy to restate it in a slightly restricted form that does not require such assumptions on what nodes know about the underlying graph.

▶ **Corollary 13.** *Protocol* Sign-Labeling *$(80\log n, 600\log n)$ performs a $(1/10)$-community-sensitive labeling, according to Definition 3 with $c_1 = 1/15$ and $c_2 = 1/6$, of any (n, d, b)-clustered regular graph G with $\lambda_3 \geqslant 1/10$ and $\lambda_2 \leqslant 1/(c\log^2 n)$ for a large enough constant c.*

Observe that the "good" time-window begins after $\mathcal{O}(n\log n)$ rounds: So, if the underlying graph has dense communities and a sparse cut, nodes can collectively compute an accurate labeling *before* the global mixing time of the graph. For instance, if the cut is $\mathcal{O}(m/n^{\gamma})$, for some constant $\gamma < 2$, our protocol is polynomially faster than the global mixing time. Importantly enough, the costs of our first protocol do not depend on the cardinality of the edge set E.

5.2 The Jump-Labeling protocol for dense cuts

The bound on the variance that allows us to adopt the sign-based criterion above does not hold when the cut is not sparse, i.e., whenever it is $\omega(m/\log^2 n)$. For such dense cuts, we use a different bound on the variance of nodes' values given in Theorem 9, which starts to hold after the global mixing time of the underlying graph and over a time window of length $\Theta(n^2)$. In this case, the specific form of the concentration bound leads to adoption of the second clustering criterion suggested by our first moment analysis, i.e., the one based on monotonicity of the values of non-ephemeral nodes. To this aim, we consider a "lazy" version of the averaging process equipped with a local clustering criterion, whereby nodes use the signs of fluctuations of their own values along consecutive rounds to label themselves (see Algorithm 3).

Here $\delta \in [0,1]$ and $\tau^{\mathrm{s}}, \tilde{\tau}^{\mathrm{s}}, \tau^{\mathrm{e}}, \tilde{\tau}^{\mathrm{e}} \in \mathbb{N}$ are parameters that will be chosen later. Intuitively, protocol Jump-Labeling exploits the expected monotonicity in the behaviour of $\mathbf{sgn}(\mathbf{x}_u^{(t)} - \mathbf{x}^{(t-1)})$ highlighted in Section 3. Though this property does not hold for a single realization of the averaging process in general, the results of Section 4.2 allow us to show that the sign of $\mathbf{x}^{(\tau_u^e)} - \mathbf{x}^{(\tau_u^s)}$ reflects u's community membership for most vertices with probability $1 - o(1)$ (i.e., the algorithm achieves weak reconstruction) when τ_u^s and τ_u^e are randomly chosen within a suitable interval. This is the intuition behind the main result of this section which is formalized below.

▶ **Theorem 14.** *Let n be any sufficiently large even positive integer. For any $0 < \delta < 0.8(\lambda_3 - \lambda_2)$, there exist $\tau^s, \tilde{\tau}^s, \tau^e, \tilde{\tau}^e \in \mathbb{N}$ such that, after $\mathcal{O}\left(\frac{n}{\delta(\lambda_3 - \lambda_2)} \log(n/\delta) + \frac{nd}{b\delta}\right)$ rounds of protocol* Jump-Labeling*$(\delta, \tau^s, \tilde{\tau}^s, \tau^e, \tilde{\tau}^e)$, every node labels its cluster and this labelling is a $\left(\sqrt[8]{\frac{\delta b}{d(\lambda_3-\lambda_2)}} + \sqrt[4]{\frac{1}{\log n}}\right)$-weak reconstruction of G, with probability at least $1 - \mathcal{O}\left(\sqrt[8]{\frac{\delta b}{d(\lambda_3-\lambda_2)}} + \sqrt[4]{\frac{1}{\log n}}\right)$. The convergence time of this algorithm is $\Omega_\delta\left(n\left(\log n + \frac{d}{b}\right)\right)$.*

Proof of Theorem 14: an informal overview. Since our discussion here will involve both local times and global times, let us define the following notation to facilitate the discussion: for each vertex $u \in V$, let $T_u : \mathbb{N} \to \mathbb{N}$ be a function that maps the local time of u to the global time, i.e., $T_u(\tau) \triangleq \min\{t \in \mathbb{N} \mid |\{i \leqslant t \mid u \in \{u_i, v_i\}\}| \geqslant \tau\}$ where $(\{u_i, v_i\})_{i \in \mathbb{N}}$ is the sequence of active edges.

We let $a_y(t) \in \mathbb{R}$ be such that $\mathbf{y}^{(t)} = a_y(t) \cdot (\chi/\sqrt{n})$. Let us also assume without loss of generality that $a_y(0) \geqslant 0$. Observe first that our concentration result (Corollaries 10 and 11) implies the following: for any t such that $\Omega(n \log n) \leqslant t \leqslant \mathcal{O}(n^2)$, with large probability, $\chi_u(\mathbf{x}_u^{(t)} - \mathbf{x}_{||,u})$ is roughly $\mathbb{E}_{\mathcal{E}}\, a_y(t)/n$ for most vertices $u \in V$; let us call these vertices *good for time t*. Imagine for a moment that we change the protocol in such a way that each u has access to the global time t and u assigns $\mathbf{h}_u^{jump} = \mathbf{sgn}(\mathbf{x}_u^{(t^{\mathrm{e}})} - \mathbf{x}_u^{(t^{\mathrm{s}})})$ for some $t^{\mathrm{s}}, t^{\mathrm{e}} \in [\Omega(n \log n), \mathcal{O}(n^2)]$ that do not depend on u. If $t^{\mathrm{e}} - t^{\mathrm{s}}$ is large enough, then $\mathbb{E}_{\mathcal{E}}\, a_y(t^{\mathrm{s}}) \gg \mathbb{E}_{\mathcal{E}}\, a_y(t^{\mathrm{e}})$. This means that, if a vertex $u \in V$ is good at both times t^{s} and t^{e}, then we have that $\chi_u(\mathbf{x}_u^{(t^{\mathrm{s}})} - \mathbf{x}_{||,u}) \approx \mathbb{E}_{\mathcal{E}}\, a_y(t^{\mathrm{s}})/n \gg \mathbb{E}_{\mathcal{E}}\, a_y(t^{\mathrm{e}})/n \approx \chi_u(\mathbf{x}_u^{(t^e)} - \mathbf{x}_{||,u})$. Note that when $\chi_u \cdot \mathbf{x}_u^{(t^{\mathrm{s}})} > \chi_u \cdot \mathbf{x}_u^{(t^{\mathrm{e}})}$, we have $\mathbf{h}_u^{jump} = \chi_u$. From this and from almost all vertices are good at both times t^{s} and t^{e}, $\mathbf{h}^{jump}$ is indeed a good weak reconstruction for the graph!

The problem of the modified protocol above is of course that, in our settings, each vertex does not know the global time t. Perhaps the simplest approach to imitate the above algorithm in this regime is to fix $\tau^{\mathrm{s}}, \tau^{\mathrm{e}} \in [\Omega(\log n), \mathcal{O}(n)]$ and, for each $u \in V$, proceed as in Jump-Labeling except with $\tau_u^{\mathrm{s}} = \tau^{\mathrm{s}}$ and $\tau_u^{\mathrm{e}} = \tau^{\mathrm{e}}$. In other words, u assigns $\mathbf{h}_u^{jump} = \mathbf{sgn}(\mathbf{x}_u^{(T_u(\tau^{\mathrm{s}}))} - \mathbf{x}_u^{(T_u(\tau^{\mathrm{e}}))})$. The problem about this approach is that, while we know that $\mathbb{E}_{\mathcal{E}}\, T_u(\tau^{\mathrm{s}}) = 0.5n\tau^{\mathrm{s}}$ and $\mathbb{E}_{\mathcal{E}}\, T_u(\tau^{\mathrm{e}}) = 0.5n\tau^{\mathrm{e}}$, the actual values of $T_u(\tau^{\mathrm{s}})$ and $T_u(\tau^{\mathrm{e}})$ differ quite a bit from their means, i.e., on average they will be $\Omega(n\sqrt{\log n})$ of away their mean. Since our concentration result only says that, at each time t, we expect 99% of the vertices to be good, it is unclear how this can rule out the following extreme case: for many $u \in V$, $T_u(\tau^{\mathrm{s}})$ or $T_u(\tau^{\mathrm{e}})$ is a time step at which u is bad. This case results in $\mathbf{h}^{jump}$ not being a good weak reconstruction of V.

The above issue motivates us to arrive at our eventual algorithm, in which τ_u^{s} and τ_u^{e} are not fixed to be the same for every u, but instead each u pick these values randomly from specified intervals $[\tau^{\mathrm{s}}, \tilde{\tau}^{\mathrm{s}}]$ and $[\tau^{\mathrm{e}}, \tilde{\tau}^{\mathrm{e}}]$. To demonstrate why this overcomes the above problem, let us focus on the interval $[\tau^{\mathrm{s}}, \tilde{\tau}^{\mathrm{s}}]$. While $T_u(\tau^{\mathrm{s}})$ and $T_u(\tilde{\tau}^{\mathrm{s}})$ can still differ from their means, the interval $[T_u(\tau^{\mathrm{s}}), T_u(\tilde{\tau}^{\mathrm{s}})]$ still, with large probability, overlaps with most of $[0.5n\tau^{\mathrm{s}}, 0.5n\tilde{\tau}^{\mathrm{s}}]$ if $\tilde{\tau}^{\mathrm{s}} - \tau^{\mathrm{s}}$ is sufficiently large. Now, if $T_u(\tau + 1) - T_u(\tau)$ are the same for all

$\tau \in [\tau^{\mathrm{s}}, \tilde{\tau}^{\mathrm{s}}]$, then the distribution of $\mathbf{x}_u^{(T_u(\tau^{\mathrm{s}}))}$ is very close to $\mathbf{x}_u^{(t_u^{\mathrm{s}})}$ if we pick t_u^{s} randomly from $[0.5n\tau^{\mathrm{s}}, 0.5n\tilde{\tau}^{\mathrm{s}}]$. From the usual global time step argument, it is easy to see that the latter distribution results in most u being good at time t_u^{s}. Of course, $T_u(\tau+1) - T_u(\tau)$ will not be the same for all $\tau \in [\tau^{\mathrm{s}}, \tilde{\tau}^{\mathrm{s}}]$, but we will be able to argue that, for almost all such τ, $T_u(\tau+1) - T_u(\tau)$ is not too small, which is sufficient for our purpose. ◀

We remark that the nd/b dependency in the running time is necessary. If we start with a good state where $\mathbf{x}^{(0)} = \mathbf{z}^{(0)} = 0$, then the values on one side of the partition are all $a_y(0)$ and the values on the other side are $-a_y(0)$. It is easy to see that, after $o(nd/b)$ steps of our protocol, $1 - o(1)$ fraction of the values remain the same. For these nodes, it is impossible to determine which cluster they are in and, hence, no good reconstruction can be achieved.

Similarly to our concentration results in Subsection 4.2, let us demonstrate the use of Theorem 14 to the two interesting cases. First, let us start with the case where $\lambda_3 - \lambda_2$ is constant.

▶ **Corollary 15.** *For any constant $\varepsilon > 0$ and for any λ_3, λ_2, there exists δ depending only on ε and $\lambda_3 - \lambda_2$ such that, for any sufficiently large n, there exists $\tau^s, \tilde{\tau}^s, \tau^e, \tilde{\tau}^e \in \mathbb{N}$ such that, with probability $1 - \varepsilon$, after $\mathcal{O}_{\varepsilon,\lambda_3-\lambda_2}\left(n \log n + \frac{n}{\lambda_2}\right)$ rounds of* Jump-Labeling*$(\delta, \tau^s, \tilde{\tau}^s, \tau^e, \tilde{\tau}^e)$, every node labels its cluster and this labelling is a ε-weak reconstruction of G.*

As in Subsection 4.2, we can consider the (non-lazy) averaging protocol and view λ_2 instead as a parameter. On this front, we arrive at the following reconstruction guarantee.

▶ **Corollary 16.** *Fix $\delta = 1/2$. For any constant $\varepsilon > 0$, any $\lambda_3 > 0.7$, any sufficiently small λ_2 depending only on ε, any sufficiently large n, there exists $\tau^s, \tilde{\tau}^s, \tau^e, \tilde{\tau}^e \in \mathbb{N}$ such that, with probability $1 - \varepsilon$, after $\mathcal{O}_{\varepsilon}\left(n \log n + \frac{n}{\lambda_2}\right)$ rounds of* Jump-Labeling*$(\delta, \tau^s, \tilde{\tau}^s, \tau^e, \tilde{\tau}^e)$, the nodes' labelling is a ε-weak reconstruction of G.*

While the weak reconstruction in the above claims is guaranteed only with arbitrarily-large constant probability, we can boost this success probability considering the same approach we used in Subsection 5.1.

Indeed, we first run $\ell = \Theta_{\varepsilon}(\log n)$ copies of Jump-Labeling where, similarly to Algorithm 2, "running ℓ copies" of Jump-Labeling means that each node keeps ℓ copies of the states of Jump-Labeling and, when an edge $\{u, v\}$ is activated, u and v jointly sample a random $j \in [\ell]$ and run the j-th copy of Jump-Labeling. In the previous section, we have seen that Lemma 8 and the repetition approach above allowed us to get a good community-sensitive labeling, w.h.p. (not a good weak-reconstruction). Interestingly enough, the somewhat stronger concentration results used in this section allow us to "add" a simple *majority* rule on the top of the ℓ components and get a "good" single-bit label, as described below. When all ℓ components of a node u have been set, node u sets $\mathbf{h}_u^{jump} = \text{Majority}_{j\in[\ell]}(\mathbf{h}_u^{jump}(i))$ where $\mathbf{h}_u^{jump}(j)$ is the binary label of u from the j-th copy of the protocol. Observe that the weak reconstruction guarantee of Jump-Labeling shown earlier implies that the expected number of mislabelings of each copy is at most $2\varepsilon n$, i.e., $\mathbb{E}[\{u \in V \mid |\mathbf{h}_u^{jump}(i) \neq \chi_u|\}] \leqslant 2\varepsilon n$. Now, since the number of mislabelings of each copy is independent, the total number of mislabelings is at most $\mathcal{O}(\varepsilon n \ell)$, w.h.p. However, if the eventual label of u is incorrect, it must contributes to mislabeling across at least $\ell/2$ copies. As a result, there are at most $\mathcal{O}(\varepsilon n)$ mislabelings in the new protocol, w.h.p.

▶ **Corollary 17.** *For any constant $\varepsilon > 0$ and $\lambda_3 > \lambda_2$, there is a protocol that yields an ε-weak reconstruction of G , w.h.p. The convergence time is $\Theta_{\varepsilon,\lambda_3-\lambda_2}\left(n\left(\log^2 n + \frac{\log n}{\lambda_2}\right)\right)$ rounds, while the work per node is $\mathcal{O}_{\varepsilon,\lambda_3-\lambda_2}\left(\log^2 n + \frac{\log n}{\lambda_2}\right)$.*

We finally remark that, for the dense-cut case we focus on in this section (i.e. $\lambda_2 = 2b/d = \Theta(1)$), the fraction of outliers turns out to be a constant we can made arbitrarily small. If we relax the condition to $\lambda_2 = o(1)$, then this fraction can be made $o(1)$, accordingly.

References

1 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

2 Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, Prasad Raghavendra, and Luca Trevisan. Average whenever you meet: Opportunistic protocols for community detection. *CoRR*, abs/1703.05045, 2017. URL: `https://arxiv.org/abs/1703.05045`.

3 Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. Find your place: Simple distributed algorithms for community detection. In *Proc. of the 28th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'17)*, pages 940–959. SIAM, 2017. `doi:10.1137/1.9781611974782.59`.

4 Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized Gossip Algorithms. *IEEE/ACM Transactions on Networking*, 14:2508–2530, 2006. `doi:10.1109/TIT.2006.874516`.

5 Gerandy Brito, Ioana Dumitriu, Shirshendu Ganguly, Christopher Hoffman, and Linh V. Tran. Recovery and Rigidity in a Regular Stochastic Block Model. In *Proc. of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 371–390. ACM, 2015.

6 Aurelien Decelle, Florent Krzakala, Cristopher Moore, and Lenka Zdeborová. Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Physical Review E*, 84(6):066106, 2011.

7 Paul W. Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

8 Laurent Massoulié. Community Detection Thresholds and the Weak Ramanujan Property. In *Proc. of the ACM Symposium on Theory of Computing (STOC)*, pages 694–703. ACM, 2014.

9 Elchanan Mossel, Joe Neeman, and Allan Sly. A proof of the block model threshold conjecture. *Combinatorica*, pages 1–44, 2013.

10 Elchanan Mossel, Joe Neeman, and Allan Sly. Belief propagation, robust reconstruction and optimal recovery of block models. In *Conference on Learning Theory*, pages 356–370, 2014.

11 Elchanan Mossel, Joe Neeman, and Allan Sly. Reconstruction and estimation in the planted partition model. *Probability Theory and Related Fields*, 162(3-4):431–461, 2015.

12 He Sun and Luca Zanetti. Distributed Graph Clustering and Sparsification. *CoRR*, abs/1711.01262, 2017. URL: `http://arxiv.org/abs/1711.01262`.

13 Matthew J. Williams, Roger M. Whitaker, and Stuart M. Allen. Decentralised detection of periodic encounter communities in opportunistic networks. *Ad Hoc Networks*, 10(8):1544–1556, 2012.

Polynomial-time approximation schemes for k-center, k-median, and capacitated vehicle routing in bounded highway dimension

Amariah Becker
Department of Computer Science, Brown University
amariah_becker@brown.edu

Philip N. Klein
Department of Computer Science, Brown University
klein@brown.edu

David Saulpic
Département d'Informatique, École Normale Supérieure
david.saulpic@ens.fr

Abstract

The concept of bounded highway dimension was developed to capture observed properties of road networks. We show that a graph of bounded highway dimension with a distinguished *root* vertex can be embedded into a graph of bounded treewidth in such a way that u-to-v distance is preserved up to an additive error of ϵ times the u-to-root plus v-to-root distances. We show that this embedding yields a PTAS for Bounded-Capacity Vehicle Routing in graphs of bounded highway dimension. In this problem, the input specifies a depot and a set of clients, each with a location and demand; the output is a set of depot-to-depot tours, where each client is visited by some tour and each tour covers at most Q units of client demand. Our PTAS can be extended to handle penalties for unvisited clients.

We extend this embedding result to handle a set S of root vertices. This result implies a PTAS for Multiple Depot Bounded-Capacity Vehicle Routing: the tours can go from one depot to another. The embedding result also implies that, for fixed k, there is a PTAS for k-Center in graphs of bounded highway dimension. In this problem, the goal is to minimize d so that there exist k vertices (the *centers*) such that every vertex is within distance d of some center. Similarly, for fixed k, there is a PTAS for k-Median in graphs of bounded highway dimension. In this problem, the goal is to minimize the sum of distances to the k centers.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Highway Dimension, Capacitated Vehicle Routing, Graph Embeddings

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.8

Related Version A full version of this article can be found at [13], arxiv.org/abs/1707.08270.

Funding Research supported by National Science Foundation grant CCF-1409520.

Acknowledgements Thanks to Andreas Feldmann and Vincent Cohen-Addad for helpful discussions and comments.

© Amariah Becker, Philip N. Klein, and David Saulpic;
licensed under Creative Commons License CC-BY

1 Introduction

The notion of *highway dimension* was introduced by Abraham et al. [3, 1] to explain the efficiency of some shortest-path heuristics. The motivation of this parameter comes from the work of Bast et al. [11, 12] who observed that, on a road network, a shortest path from a compact region to points that are far enough must go through one of a small number of nodes. They experimentally showed that the US road network has this property, and Abraham et al. [3, 1, 2] proved results on the efficiency of shortest-path heuristics on graphs with bounded highway dimension.

Though several definitions of highway dimension have been proposed, we use the one given in [20] :

▶ **Definition 1.** The *highway dimension* of a graph $G = (V, E)$ is the smallest integer η such that for every $r \in \mathbb{R}^+$ and $v \in V$, there is a set of at most η vertices in $B_v(cr)$ such that every shortest path of length at least r that has all its vertices in $B_v(cr)$ intersects this set.

$B_v(r) = \{u \in V | d(u, v) \leq r\}$ denotes here the ball with center v and radius r. This definition is chosen as it captures this property for a wider range of transportation networks than [2]. Since the latter implies low doubling dimension, it cannot, for example, represent air traffic networks, that are star-like at large airports which causes a large doubling dimension. Nevertheless, as noted in Feldman et al. [20], these networks have a low highway dimension according to the definition of this paper (see the full version for a further discussion of these definitions).

New polynomial-time approximation schemes: Abraham et al. note that "conceivably, better algorithms for other [optimization] problems can be developed and analyzed under the small highway dimension assumption." Since road networks are thought to be modeled by graphs of small highway dimension, NP-hard optimization problems that arise in road networks are natural candidates for study. Feldmann [19] and Feldmann, Fung, Könemann, and Post [20] inaugurated this line of research, giving (respectively) a constant-factor approximation algorithm for one problem and quasi-polynomial-time approximation schemes for several other problems. In this paper, we give the first *polynomial-time approximation schemes* (PTASs) for classical optimization problems in graphs of small highway dimension.

Vehicle routing: Consider Capacitated Vehicle Routing, defined as follows. An instance consists of a positive integer Q (the *capacity*), a graph with edge-lengths, a subset Z of vertices (called *clients*), a demand function $\rho : Z \to \{1, 2 \ldots, Q\}$, and a distinguished vertex, called the *depot*. A solution consists of a set of *tours*, where each tour is a walk that starts and ends at the depot, and a function that assigns each client to a tour that passes through it, such that the total client demand assigned to each tour is at most Q. (If a client v is assigned to a tour, we say that the tour *visits* v.) The objective is to minimize the sum of lengths of the tours.

We emphasize that in this version of Capacitated Vehicle Routing, client demand is *indivisible*: a client's entire demand must be covered by a single tour. For arbitrary metrics, the problem is APX-hard, even when $Q > 0$ is fixed [9]. When Q is unbounded, it is NP-hard to approximate to within a factor of 1.5 even when the metric is that of a star [21]. Since stars have highway dimension one, this hardness result holds for graphs of bounded highway dimension. We therefore require Q to be constant. To emphasize this, we sometimes refer to the problem as Bounded-Capacity Vehicle Routing.

▶ **Theorem 2.** *For any $\epsilon > 0$, $\eta > 0$ and $Q > 0$, there is a polynomial-time algorithm that, given an instance of* Bounded-Capacity Vehicle Routing *in which the capacity is Q and the graph has highway dimension η, finds a solution whose cost is at most $1 + \epsilon$ times optimum.*

The running time is bounded by a polynomial whose degree depends on ε, η, and Q. PTASs for vehicle routing were previously known only for Euclidean spaces, although a quasi-polynomial-time approximation scheme (QPTAS) was known for planar graphs (see Section 1.2).

Our approach can be modified to handle a generalization in which an instance also specifies a *penalty* for each client, to be imposed if the solution omits the client. We also give a PTAS for a more general version of the problem, Multiple-Depot Bounded-Capacity Vehicle Routing, in which there are a constant number of depots, and each tour is required only to start and end at one of the depots.

k-Center and k-Median: Given a graph, the goal in k-Center is to select a set of k vertices (the *centers*) so as to minimize the maximum distance of a vertex to the nearest center. This problem might arise, for example, in selecting locations for k firehouses. The objective in k-Median is to minimize the average vertex-to-center distance.

For k-Center, when the number k of centers is unbounded, for any $\delta > 0$, it is NP-hard [22, 28] to obtain a $(2 - \delta)$-approximation, even in the Euclidean plane under L_1 or L_∞ metrics[1], even in unweighted planar graphs [31], and even in n-vertex graphs with highway dimension $O(\log^2 n)$ [19]. We therefore consider bounded k, but even a $(2 - \epsilon)$-approximation is $W[2]$-hard for parameter k [19] in general graphs. Thus, even for bounded k, it seems necessary to consider restricted inputs. Feldmann [19] gave a polynomial-time $3/2$-approximation algorithm for bounded-highway-dimension graphs, and raised the question of whether a better approximation ratio could be achieved. The following theorem answers that question (Note that the running time is bounded by a polynomial in n whose degree does *not* depend on η, k, or ε).

▶ **Theorem 3.** *There is a function $f_1(\cdot, \cdot, \cdot)$ and a constant c such that, for each of the problems k-*Center *and k-*Median*, for any $\eta > 0$, $k > 0$ and $\varepsilon > 0$, there is an algorithm running in time $f_1(\eta, k, \varepsilon)n^c$ that, given an instance in which the graph has highway dimension at most η, finds a solution whose cost is at most $1 + \varepsilon$ times optimum.*

1.1 New metric embedding results

The key to achieving the new approximation schemes is a new result on metric embeddings of bounded-highway-dimension graphs into bounded-treewidth graphs. *Treewidth* is a measure of how complicated a graph is, and many NP-hard optimization problems in graphs become polynomial-time solvable when the input is restricted to graphs of bounded treewidth. The definition is the following.

A *tree decomposition* of a graph G is a tree T_G whose nodes are *bags* of vertices that satisfy the following three criteria: every $v \in V$ appears in at least one bag, for every edge $(u, v) \in E$ there is some bag containing both u and v and for every $v \in V$, the bags containing v form a connected subtree. The *width* of T_G is the size of the largest bag minus one, and the *treewidth* of G is the minimum width among all tree decompositions of G.

[1] Approximation better than 1.822 is hard under L_2, see [18].

A metric embedding of an (undirected) guest graph G into a host graph H is a mapping $\phi(\cdot)$ from the vertices of G to the vertices of H such that, for every pair of vertices u, v in G, the $\phi(u)$-to-$\phi(v)$ distance in H resembles the u-to-v distance in G. Usually in studying metric embeddings one seeks an embedding that preserves u-to-v distance up to some factor (the *distortion*). That is, the allowed error is proportional to the original distance. In this work, the allowed error is instead proportional to the distance from a given root vertex (or a constant number of vertices).

▶ **Theorem 4.** *There is a function $f_2(\cdot,\cdot)$ such that, for every $\varepsilon > 0$, graph G of highway dimension η, and vertex s, there exists a graph H and an embedding $\phi(\cdot)$ of G into H such that*

- *H has treewidth at most $f_2(\varepsilon, \eta)$, and*
- *for all vertices u and v, $d_G(u,v) \leq d_H(\phi(u),\phi(v)) \leq d_G(u,v) + \varepsilon(d_G(s,u) + d_G(s,v))$.*

As we describe in greater detail in Section 5, our PTAS for Bounded-Capacity Vehicle Routing first applies Theorem 4 with s being the depot and $\epsilon' = \epsilon/c$ for a constant c to be determined, obtaining an embedding of the original graph into the bounded-treewidth graph H. The embedding induces an instance of Vehicle Routing in H. The algorithm finds an optimal solution to this instance, and converts it to a solution for the original instance. This conversion does not increase the cost of the solution. However, we need to show that the optimal solution in the original instance induces a solution in H of not too much greater cost. We do this using a lower bound due to Haimovich and Rinnoy Kan [26].

For the multiple-depot version of vehicle routing and for k-Center and k-Median, Theorem 4 does not suffice. We present a generalization in which there is a set of root vertices, and the allowed error is proportional to the minimum distance to any root vertex.

▶ **Theorem 5.** *There is a function $f_3(\cdot,\cdot,\cdot)$ such that, for every $\varepsilon > 0$, graph G of highway dimension η and set S of vertices of G, there exists a graph H and an embedding $\phi(\cdot)$ of G into H such that*

- *H has treewidth $f_3(\eta, |S|, \varepsilon)$, and*
- *for all u and v, $d_G(u,v) \leq d_H(\phi(u),\phi(v)) \leq (1+O(\varepsilon))d_G(u,v) + \varepsilon \min(d_G(S,u), d_G(S,v))$*

1.2 Related Work

Metric embeddings of bounded-highway-dimension graphs: Feldmann [19] and Feldmann et al. [20] inaugurated research into approximation algorithms for NP-hard problems in bounded-highway-dimension graphs. Feldmann et al. [20] gave quasi-polynomial-time approximation schemes for Traveling Salesman, Steiner Tree, and Facility Location. The key to their results is a probabilistic metric embedding of bounded-highway dimension graphs into graphs of small treewidth. The *aspect ratio* of a graph with edge-lengths is the ratio of the maximum vertex-to-vertex distance to the minimum vertex-to-vertex distance. Feldmann et al. show that, for any $\epsilon > 0$, for any graph G of highway dimension η, there is a probabilistic embedding $\phi(\cdot)$ of G of expected distortion $1 + \epsilon$ into a randomly chosen graph H whose treewidth is polylogarithmic in the aspect ratio of G (and also depends on ϵ and η). There are two obstacles to using this embedding in achieving approximation schemes:

- The distortion is achieved only in expectation. That is, for each pair u, v of vertices, the expected $\phi(u)$-to-$\phi(v)$ distance in H is at most $(1 + \epsilon)$ times the u-to-v distance in G.
- The treewidth depends on the aspect ratio of G, so is only bounded if the aspect ratio is bounded.

The first is an obstacle for problems (e.g. k-CENTER) where individual distances need to be bounded; this does not apply to problems such as TRAVELING SALESMAN or VEHICLE ROUTING where the objective is a sum of lengths of paths. The second is the reason that Feldmann et al. obtain only quasi-polynomial-time approximation schemes; it seems to be an obstacle to obtaining true PTAS. Nevertheless, the techniques introduced by Feldmann et al. are at the core of our embedding results. We build heavily on their framework.

About VEHICLE ROUTING PROBLEM, Haimovich and Rinnoy Kan [26] proved the following lower bound[2]:

▶ **Lemma 6.** *For* CAPACITATED VEHICLE ROUTING *with capacity Q, and client set Z,*

$$cost(OPT) \geq \frac{2}{Q}\sum\{d(c,s) \ : \ c \in Z\}$$

Note that the CAPACITATED VEHICLE ROUTING problem is a generalization of TRAVELING SALESMAN ($Q = n$, $Z = V$, and $\rho(v) = 1, \forall v$). Conversely, Haimovich and Rinnoy Kan show how to use a solution to TRAVELING SALESMAN to achieve a constant-factor approximation for CAPACITATED VEHICLE ROUTING, where the constant depends on the approximation ratio for TRAVELING SALESMAN.

Since CAPACITATED VEHICLE ROUTING in general graphs is APX-hard for every fixed $Q \geq 3$ [8, 9], much work has focused on the Euclidean plane. Haimovich and Rinnoy Kan [26] gave a polynomial-time approximation scheme (PTAS) for the Euclidean plane for the case when the capacity Q is constant. Asano et al. [9] showed how to improve this algorithm to get a PTAS when Q is $O(\log n/\log\log n)$. For general capacities, Das and Mathieu [17] gave a quasi-polynomial-time approximation scheme for unbounded Q. Building on this work, Adamaszek, Czumaj, and Lingas [4] gave a PTAS that for any $\epsilon > 0$ can handle Q up to $2^{\log^\delta n}$ where δ depends on ϵ.

Little is known for higher dimensions or other metrics. Kachay gave a PTAS in $\mathbb{R}^d$ that requires Q to be $O(\log^{1/d}\log n)$ [30], and Hamaguchi and Katoh [27] and Asano, Katoh, and Kawashima [7] focused on constant-factor approximation algorithms for the case where the graph is a tree and client demand is divisible. Becker, Klein and Saulpic [14] gave the first approximation scheme for a non-Euclidean metric: they describe a quasi-polynomial-time approximation scheme in planar graphs, but only when the capacity Q is polylogarithmic in the graph size. They introduce the idea of an error that depends on the distance to the depot, which we also use in the embedding presented in our work here.

For k-MEDIAN, constant-factor approximation algorithms have been found for general metric spaces [15, 32, 29, 6]. The best known approximation ratio for k-MEDIAN in general metrics is 2.675 [15], and it is NP-hard to approximate within a factor of $1 + 2/e$ [23]. For k-MEDIAN in d-dimensional Euclidean space, PTAS have been found when k is fixed (e.g. [10]) and when d is fixed (e.g. [5]) but there exists no PTAS if k and d are part of the input [25]. Recently Cohen-Addad et al. [16] gave a local search-based PTAS for k-MEDIAN in edge-weighted planar graphs, and more generally in graphs from any nontrivial minor-closed graph family.

Outline. Section 2 provides preliminary definitions and presents useful results from Feldmann et al. [20]. In Section 3 we give an initial embedding result for graphs of bounded aspect ratio. Section 4 explains the main embedding result (Theorem 4), and Section 5 describes

[2] Although their result addresses the unit-demand case, it generalizes to instances where each non-zero client demand $\rho(v)$ is at least one.

how to use this embedding to achieve a PTAS for Capacitated Vehicle Routing, proving Theorem 2. We refer the reader to the full version [13] for a discussion of highway dimension, omitted proofs, the dynamic program for vehicle routing, and a discussion of Theorem 5 and its application to multi-depot vehicle routing, k-Center, and k- Median.

2 Preliminaries

We use OPT to denote the optimum solution for an optimization problem. For minimization problems, an α-approximation algorithm returns a solution with cost at most $\alpha \cdot \text{cost}(OPT)$. An *approximation scheme* is a family of $(1+\varepsilon)$-approximation algorithms indexed by $\varepsilon > 0$. A *polynomial-time approximation scheme* (PTAS) is an approximation scheme that for each fixed ε runs in polynomial time.

For an undirected graph $G = (V, E)$, we use $d_G(u, v)$ (or $d(u, v)$ when G is unambiguous) to denote the shortest-path distance between u and v. For any vertex subsets $W \subseteq V$ and vertex $v \in V$ we let $d(v, W)$ denote $\min_{w \in W} d(v, w)$, and we let $diam(W)$ denote $\max_{u,v \in W} d(u, v)$.

An *embedding* of a graph $G = (V, E)$ is a mapping ϕ from a *guest* graph G to a *host* graph $H = (V, E_H)$. For notational simplicity, we identify the vertices of H with points of G and therefore omit ϕ.

Let $Y \subseteq X$ be a subset of elements in a metric space (X, d). Y is a δ*-covering* of X if for all $x \in X$, $d(x, Y) \leq \delta$. Y is a β*-packing* of X if for all $y_1, y_2 \in Y$ with $y_1 \neq y_2$, $d(y_1, y_2) \geq \beta$. Y is an ε*-net* if it is both an ε-covering and an ε-packing.

Shortest-Path Covers. Now we introduce a tool for dealing with bounded highway-dimension graphs. Recall that c is a constant greater than 4.

▶ **Definition 7.** For a graph G with vertex set V and $r \in R^+$, a *shortest-path cover for scale* r SPC$(r) \subseteq V$ is a set of vertices, called *hubs*, such that every shortest path of length in $(r,\ cr/2]$ contains at least one hub. Such a cover is called *locally s-sparse* for scale r if every ball of diameter cr contains at most s vertices from SPC(r).

For a graph of highway dimension η, Abraham et al. [1] showed how to find a locally $O(\eta \log \eta)$-sparse shortest-path cover in polynomial time (though they show it for a different definition of highway dimension ($c = 4$), the algorithm can be straightforwardly adapted). This result allows us to use shortest-path covers instead of directly using highway dimension.

Town Decomposition. Feldmann et al.[20] observed that a shortest-path cover for scale r naturally defines a clustering of the vertices into *towns* [20]. Informally, a *town* at scale r is a subset of vertices that are close to each other and far from other towns and from the shortest-path cover for scale r. Formally, a town is defined by at least one $v \in V$ such that $d(v, \text{SPC}(r)) > 2r$ and is composed of $\{u \in V | d(u, v) \leq r\}$. The following lemma of Feldmann et al. describes key properties of towns.

▶ **Lemma 8** (Lemma 3.2 in [20]). *If T is a town at scale r, then*
1. *$diam(T) \leq r$ and*
2. *$d(T, V \setminus T) > r$*

Feldmann et al. define a recursive decomposition of the graph using the concept of towns, which we adopt for this paper. First, scale all distances so that the shortest point-to-point distance is a little more than $c/2$. Then fix a set of scales $r_i = (c/4)^i$. We say that a town

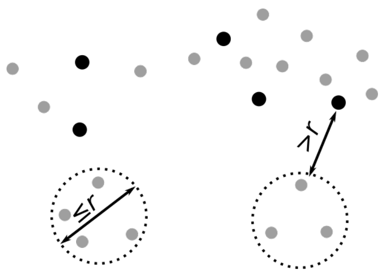


Figure 1 Illustration of Lemma 8.

T at scale r_i is on *level* i. The scaling ensures that $SPC(r_0) = \emptyset$, and therefore at level 0 every vertex forms a singleton town. The largest level is $r_{max} = \lceil \log_{c/4} diam(G_{scaled}) \rceil = \lceil \log_{c/4}(\frac{c}{2} \cdot \theta_G)) \rceil$, where θ_G is the aspect ratio of the input graph. Similarly at this topmost level, $SPC(r_{max}) = \emptyset$ since there are no shortest paths that need to be covered. The only town at scale r_{max} is the town that contains the entire graph. We say that the town at scale r_{max} and the singleton towns at scale r_0 are *trivial* towns. Since c is a constant greater than four, the total number of scales is linear in the input size.

The set $\mathcal{T} = \{T \subseteq V | T$ is a town on level $i \in \mathbb{N}\}$ of towns at all levels is called the *town decomposition*. Because of the properties of Lemma 8, this set forms a laminar family and therefore has a tree structure. Moreover, the decomposition has the following properties.

▶ **Lemma 9** (Lemma 3.3 in [20]). *For every town T in a town decomposition $\mathcal{T}$,*
1. *T has either 0 children or at least 2 children, and*
2. *if T is a town at level i and has child town T' at level j, then $j < i$.*

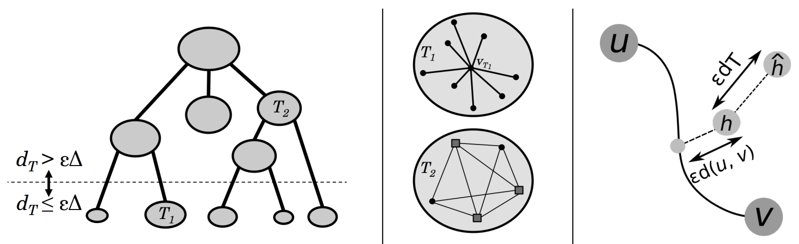
Approximate Core Hubs. For the purpose of approximation algorithms, it suffices to use not all hubs but a representative subset. For $\varepsilon > 0$, Feldmann et al. show how to compute, for each town T, a subset X_T of $T \bigcap \cup_i \text{SPC}(r_i)$, called *approximate core hubs*. Their properties are described in Lemma 10. Recall that the *doubling dimension* of a metric is the smallest θ such that for every r, every ball of radius $2r$ can be covered by at most 2^θ balls of radius r.

▶ **Lemma 10** (Theorem 4.2 and Lemma 5.1 in [20]). *For every town $T \in \mathcal{T}$, there exist a set X_T such that:*
1. *if T_1 and T_2 are different child towns of T, and $u \in T_1$ and $v \in T_2$, then there is some $h \in X_T$ such that $d(P[u,v], h) \leq \varepsilon d(u,v)$, where $P[u,v]$ is the shortest u-to-v path, and*
2. *the doubling dimension of X_T is $\theta = O(\log(\eta s \log(1/\varepsilon))$.*

Minimality of Shortest-Path Covers. Note that the result of Lemma 10 requires the shortest-path covers be inclusion-wise minimal. For the embedding we present in Section 4, however, it is useful to assume that the depot is not a member of any town except for the trivial topmost town containing all of G and bottommost singleton town containing just the depot. This assumption can be made safely, as explained in the full version of the paper.

3 Embedding for Graphs of Bounded Aspect-Ratio

Lemma 11 describes an embedding for the case when the graph has bounded aspect-ratio, ie. the ratio between diameter and smallest distance. This embedding gives only a small *additive* error, and will prove to be a useful tool for the following sections. In this section we show how to construct this embedding.



(a) Town decomposition **(b)** Embeddings **(c)** Path approximation

Figure 2 (a) An example of a town decomposition. T_1 has diameter at most $\varepsilon\Delta$ and T_2 has diameter greater than $\varepsilon\Delta$. (b) Two cases of town embeddings. T_1 is embedded as a star with center v_{T_1}. The embedding of T_2 connects all vertices in T_2 to all hubs in $\hat{X}_{T_2}$(depicted as squares). (c) Hub $\hat{h} \in \hat{X}_T$ is close to hub $h \in X_T$ which itself is close to the shortest u-to-v path.

▶ **Lemma 11.** *There is a function $f(x, y)$ such that, for any $\varepsilon > 0$ and $\eta > 0$, for any graph G with highway dimension at most η, minimal distance 1 and diameter Δ, there is a graph H with treewidth at most $f(\varepsilon, \eta)$ and an embedding $\phi(\cdot)$ of G into H such that, for all points u and v,*

$$d_G(u, v) \leq d_H(\phi(u), \phi(v)) \leq d_G(u, v) + 4\varepsilon\Delta$$

Furthermore, there is a polynomial-time algorithm to construct H and the embedding.

We first present an algorithm to compute the host graph H and a tree decomposition of H. This algorithm relies on the town decomposition $\mathcal{T}$ of G, described in Section 2.

The host graph H is constructed as follows. First, consider a town T that has diameter $d \leq \varepsilon\Delta$ but has no ancestor towns of diameter $\varepsilon\Delta$ or smaller. We call such a town a *maximal* town of diameter at most $\varepsilon\Delta$. The town T is embedded into a star: choose an arbitrary vertex v_T in T, and for each $u \in T$, include an edge in H between u and v_T with length $d_G(u, v_T)$ equal to their distance in G (see Figures 2a and 2b).

Now consider a town T of diameter $d_T > \varepsilon\Delta$. The set of approximate core hubs X_T can be used as *portals* to preserve distances between vertices lying in different child towns of T. Specifically, by Lemma 10, for every pair of vertices (u, v) in different child towns of T, X_T contains a vertex that is close to the shortest path between u and v. In order to approximate the shortest paths, it is therefore sufficient to consider a set of points *close to* X_T. Let $\hat{X}_T$ be an εd_T-net of X_T. For each $\hat{h} \in \hat{X}_T$ and $v \in T$, include an edge in H connecting v to $\hat{h}$ with length $d_H(v, \hat{h}) = d_G(v, \hat{h})$ equal to the v-to-$\hat{h}$ distance in G (see Figures 2a and 2b).

The tree decomposition D mimics the town decomposition tree: for each town T of diameter greater than $\varepsilon\Delta$, there is a bag b_T. This bag is connected in D to all of the bags of child towns of T and contains all of the vertices of the net assigned to T and of the nets assigned to T's ancestors in the town decomposition. Formally, if A_T denotes the set of all towns that contain T, $b_T = \bigcup_{T' \in A_T} \hat{X}_{T'}$. Note that if T' is the parent of T in the town decomposition, $b_T = \hat{X}_T \cup b_{T'}$. Now for each maximal town T of diameter at most $\varepsilon\Delta$ with parent town T', the tree decomposition contains a bag b_T^0 connected to a bag b_T^u for each vertex $u \in T$. We define $b_T^0 = \{v_T\} \cup b_{T'}$ and $b_T^u = \{u\} \cup b_T^0$.

Following Feldmann et al. [20], the above construction can be shown to be polynomial-time constructible. The following three lemmas therefore prove Lemma 11.

▶ **Lemma 12.** *D is a valid tree decomposition of H.*

▶ **Lemma 13.** *H has a treewidth $O((\frac{1}{\varepsilon})^\theta \log_{\frac{c}{4}} \frac{1}{\varepsilon})$, where θ is a bound on the doubling dimension of the sets X_T.*

Proof. Since the size of the bags is clearly bounded by the depth times the maximal cardinality of $\hat{X}_T$, it is enough to prove that, for each town T, $\hat{X}_T$ is bounded by $(\frac{1}{\varepsilon})^\theta$, and that the tree decomposition has a depth $O(\log_{\frac{c}{4}} \frac{1}{\varepsilon})$. By Lemma 10, the doubling dimension of X_T is bounded by θ. $\hat{X}_T$ is a subset of X_T, so its doubling dimension is bounded by 2θ (see Gupta et al. [24]). Furthermore, the aspect ratio of $\hat{X}_T$ is $\frac{1}{\varepsilon}$: the longest distance between members of $\hat{X}_T$ is bounded by the diameter d_T of the town, and the smallest distance is at least εd_T by definition of a net. The cardinality of a set with doubling dimension x and aspect ratio γ is bounded by $2^{x\lceil \log_2 \gamma \rceil}$ (see [24] for a proof), therefore $|\hat{X}_T|$ is bounded by $(\frac{1}{\varepsilon})^\theta$. We prove now that the tree decomposition has a depth $O(\log_{\frac{c}{4}} \frac{1}{\varepsilon})$. Let T be a town of diameter $d_T > \varepsilon\Delta$ and let r_i be the scale of that town. By Lemma 8, $d_T \leq r_i$, and since $r_i = (\frac{c}{4})^i$ and $d_T > \varepsilon\Delta$, we can conclude that $i > \log_{\frac{c}{4}} \varepsilon\Delta$. As the diameter of the graph is Δ, the biggest town has a diameter at most Δ. It follows that $r_i \leq \Delta$ and therefore $i \leq \log_{\frac{c}{4}} \Delta$. The depth of b_T in the tree decomposition is therefore bounded by $\log_{\frac{c}{4}} \frac{\Delta}{\varepsilon\Delta} = \log_{\frac{c}{4}} \frac{1}{\varepsilon}$. Furthermore, the tree decomposition of a town of diameter at most $\varepsilon\Delta$ has depth 2. The overall depth is therefore $O(\log_{\frac{c}{4}} \frac{1}{\varepsilon})$, concluding the proof. ◀

▶ **Lemma 14.** *For all vertices u and v, $d_G(u, v) \leq d_H(u, v) \leq d_G(u, v) + 4\varepsilon\Delta$*

Proof. Let u and v be vertices in V, and let T be the town that contains both u and v such that u and v are in different child towns of T.

If T has diameter $d_T \leq \varepsilon\Delta$, then let T' be the maximal town of diameter at most $\varepsilon\Delta$ that is an ancestor of T (possibly T itself). By construction, T' was embedded into a star centered at some vertex $v_{T'} \in T'$, so $d_H(u, v) \leq d_H(u, v_{T'}) + d_H(v_{T'}, v) \leq d_G(u, v_{T'}) + d_G(v_{T'}, v) \leq 2\varepsilon\Delta$.

Otherwise if T has diameter $d_T > \varepsilon\Delta$, then by Lemma 10, there is some $h \in X_T$ such that $d_G(P[u, v], h) \leq \varepsilon d(u, v)$. Since $\hat{X}_T$ is an εd_T cover of X_T, there is some $\hat{h} \in \hat{X}_T$ such that $d(h, \hat{h}) \leq \varepsilon d_T$. The host graph H includes edges $(u, \hat{h})$ and $(\hat{h}, v)$, so

$$d_H(u, v) \leq d_H(u, \hat{h}) + d_H(\hat{h}, v) \leq d_G(u, h) + d_G(h, v) + 2\varepsilon d(u, v) + 2\varepsilon d_T \leq d_G(u, v) + 4\varepsilon\Delta$$

(see Figure 2c). Finally, since edge lengths in H are given by distances in G, $d_G(u, v) \leq d_H(u, v)$ for all $u, v \in V$. ◀

4 Main Embedding: Proof of Theorem 4

4.1 Embedding Construction

Given the parameter $\hat{\epsilon}$, our goal for the embedding is that

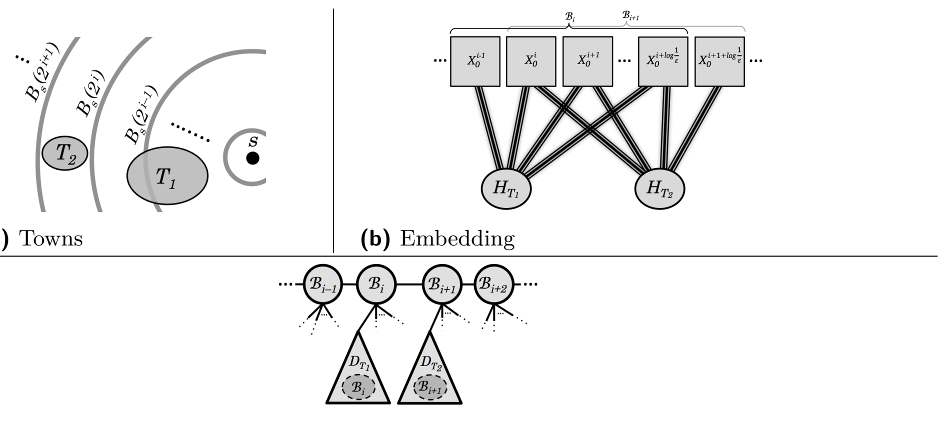
$$d_G(u, v) \leq d_H(\phi(u), \phi(v)) \leq d_G(u, v) + \hat{\varepsilon}(d_G(s, u) + d_G(s, v))$$

With this goal in mind, we define $\epsilon = \min\{1/4, \hat{\varepsilon}/k\}$ for an appropriate constant k (chosen to compensate for the big-O in the following inequality), and prove that

$$d_G(u, v) \leq d_H(\phi(u), \phi(v)) \leq d_G(u, v) + \varepsilon(d_G(s, u) + d_G(s, v))$$

Our construction relies on the assumption that the depot s does not appear in any non-trivial town. We can make this assumption without loss of generality, as discussed in Section 2.

The root town in the composition, denoted T_0, is the town that contains the entire graph. We say that a town T that is a child of the root town is a *top-level town*, which means that the only town that properly contains T is T_0.



(a) Towns **(b)** Embedding

(c) Tree decomposition

Figure 3 (a) Towns T_1 and T_2 are top-level towns, with $l(T_1) = i$ and $l(T_2) = i + 1$. (b) The embedding of each top-level town (circles) are connected to a band of $\log_2 \frac{1}{\varepsilon} + 1$ hub sets (squares). Edges are striped to convey that they connect *all* vertices of the given hub-set endpoint to *all* vertices of the town-embedding endpoint. (c) The vertices of each bag $\mathcal{B}$ (circles) are added to each bag of each descendent top-level-town tree decomposition (triangles).

The assumption that the depot, s, does not appear in any non-trivial town implies that the top-level town that contains s is the trivial singleton town. This assumption is helpful to bound the distance between a top-level town T and the depot s: as $s \notin T$, Lemma 8 gives the bound $d(T, s) \geq \operatorname{diam}(T)$. This bound turns out to be very helpful in the construction of the host graph.

We use Lemma 11 to construct an embedding for each top-level town. It remains to connect these embeddings : we cannot approximate X_{T_0} with a net as we did in Lemma 11, because the diameter of G may be arbitrarily large.

To cope with that issue, we define inductively the hub sets $X_0^0, X_0^1, \ldots$ such that X_0^k is a net of $X_{T_0} \cap B_s(2^k)$. Let X_0^0 be an ε-net of $X_{T_0} \cap B_s(1)$ that contains the depot, s, and for $k \geq 0$ let X_0^{k+1} be an $\varepsilon 2^{k+1}$-net of the set $\big(X_{T_0} \cap (B_s(2^{k+1}) - B_s(2^k))\big) \bigcup X_0^k$ that contains the depot. This construction ensures that $X_0^{k+1} \cap B_s(2^k) \subseteq X_0^k$, which will be helpful in Section 4.3 to find a tree decomposition of the host graph. Note that we can assume $s \in X_{T_0}$, since adding it increases the doubling dimension by at most one and thus does not change the result of Lemma 10.

For a set of vertices $\mathcal{X} \subseteq V$, we define $l(\mathcal{X}) = \lceil \log_2(\max_{v \in \mathcal{X}} d(s, v)) \rceil$ (See Figure 3a).

For every child town T of T_0, the host graph connects every vertex v of T to every hub h in $X_0^{l(T)}, \ldots, X_0^{l(T)+\log_2(1/\varepsilon)}$ with an edge of length $d_G(v, h)$ (See Figure 3b).

4.2 Proof of Error Bound

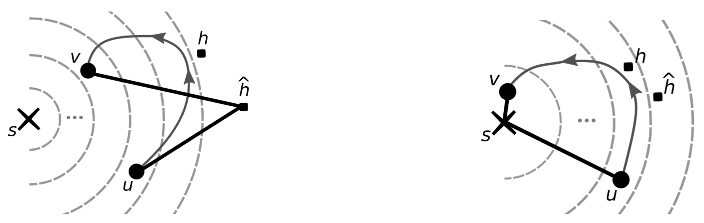
In Lemma 16 we prove a bound on the error incurred by the embedding. Our proof makes use of the following lemma.

▶ **Lemma 15** (see full version). *For all k, X_0^k is an $\varepsilon 2^{k+1}$-covering of $X_{T_0} \cap B_s(2^k)$.*

▶ **Lemma 16.** *For all vertices u and v,*

$$d_G(u, v) \leq d_H(u, v) \leq d_G(u, v) + O(\varepsilon)(d_G(s, u) + d_G(s, v))$$

Proof. Consider two vertices u and v. Let T_u and T_v denote the top-level towns that contain u and v, respectively. There are two cases to consider.



(a) u and v are both connected to $\hat{h}$. **(b)** v is not connected to $\hat{h}$.

Figure 4 The shortest path between u and v in G is indicated by the curved, directed lines. The path in the host graph is represented by the straight lines.

If $T_u = T_v$, Lemma 8 gives $d_G(u,v) \leq diam(T_u) \leq d_G(T_u, V \setminus T_u)$, and therefore $diam(T_u) \leq \min\{d_G(s,u),\ d_G(s,v)\}$. Because $T_u = T_v$ is a top-level town, its embedding is given by Lemma 11, which directly gives the desired bound.

Otherwise $T_u \neq T_v$. Without loss of generality, assume that $d_G(u,s) \geq d_G(v,s)$. We show that there exists some X_0^k connected to u with a vertex $\hat{h} \in X_0^k$ close to $P[u,v]$.

By definition of the approximate core hubs, there exists $h \in X_{T_0}$ such that $d(h, P[u,v]) \leq \varepsilon d(u,v)$. Moreover, $h \in B_s(2^{l(T_u)+2})$:

$$\begin{aligned} d(s,h) &\leq d(s,u) + d(u,h) \leq d(s,u) + (1+\varepsilon) d(u,v) \\ &\leq d(s,u) + (1+\varepsilon)\left(d(s,u) + d(s,v)\right) \qquad \text{by triangle inequality} \\ &\leq d(s,u) + (1+\varepsilon) \cdot 2d(s,u) \qquad \text{since } d(u,s) \geq d(v,s) \\ &\leq (3+2\varepsilon)2^{l(T_u)} \leq 2^{l(T_u)+2} \end{aligned}$$

Since $h \in X_{T_0} \cap B_s(2^{l(T_u)+2})$, then by Lemma 15, there is an $\hat{h} \in X_0^{l(T_u)+2}$ such that $d(\hat{h}, h) \leq \varepsilon 2^{l(T_u)+3}$. Since $\log_2 \frac{1}{\varepsilon} \geq 2$, u is connected to $\hat{h}$ in the host graph.

Depending on v, there remain two cases: either v is connected to $\hat{h}$ (see Figure 4a) or not (Figure 4b). First, if v is connected to $\hat{h}$ in the host graph, $d_H(v,\hat{h}) = d_G(v,\hat{h})$ (and the same holds for u). The triangle inequality gives therefore,

$$d_H(u,v) \leq d_G(u,\hat{h}) + d_G(v,\hat{h}) \leq \underbrace{d_G(u,h) + d_G(v,h)}_{\leq (1+2\varepsilon)d_G(u,v) \text{ by definition of } h} + \underbrace{2d_G(\hat{h},h)}_{\leq 2\varepsilon 2^{l(T_u)+3} = O(\varepsilon)d(s,u)}$$

Since $d_G(u,v) \leq d_G(s,u) + d_G(s,v)$, we infer $d_H(u,v) \leq d_G(u,v) + O(\varepsilon)(d_G(s,u) + d_G(s,v))$.

Otherwise, v is not connected to $\hat{h}$. That means that either $l(T_u) + 2 < l(T_v)$ or $l(T_u) + 2 > l(T_v) + \log_2 \frac{1}{\varepsilon}$. We exclude the first case by noting that since the diameter of a town is less than its distance to the depot, $d_G(v,s) \leq d_G(u,s)$ implies that $l(T_v) \leq l(T_u) + 1$. The second case implies that $d_G(s,u) \geq O(\frac{1}{\varepsilon}) d_G(s,v)$. Since the host graph connects the source s to all the vertices, $d_H(u,v) \leq d_G(s,u) + d_G(s,v) \leq d_G(u,v) + 2d_G(s,v) \leq d_G(u,v) + O(\varepsilon)(d_G(s,u) + d_G(s,v))$. ◀

4.3 Tree Decomposition

We present here the construction of a bounded-width tree decomposition D of the host graph.

For each $k > 0$ let $\mathcal{B}_k = \bigcup_{i=k-1}^{k+\log_2(1/\varepsilon)} X_0^i$. For a top-level town T, the tree decomposition D connects the decomposition D_T given by Lemma 11 to the bag $\mathcal{B}_{l(T)}$. Moreover, we add all vertices that appear in $\mathcal{B}_{l(T)}$ to all bags in the tree D_T. Finally, for every k we connect $\mathcal{B}_k$ to both $\mathcal{B}_{k-1}$ and $\mathcal{B}_{k+1}$ in D. (See Figure 3b.)

▶ **Lemma 17** (see full version). *D is a valid tree decomposition of the host graph H.*

▶ **Lemma 18.** *For all k, $|X_0^k| \leq (\frac{2}{\varepsilon})^\theta$.*

Proof. Since X_0^k is a subset of X_{T_0}, it has doubling dimension 2θ (see Lemma 10). Since X_0^k is a $\varepsilon 2^k$-net, the smallest distance between two hubs in X_0^k is at least $\varepsilon 2^k$. Moreover, since $X_0^k \subseteq B_s(2^k)$, the longest distance between two hubs is at most $2 \cdot 2^k$, therefore, X_0^k has an aspect ratio of at most $\frac{2}{\varepsilon}$. The bound used in Lemma 13 on the cardinality of a set using its aspect ratio and its doubling dimension concludes the proof. ◀

▶ **Lemma 19.** *The tree decomposition D has bounded width.*

Proof. This follows from Lemma 18 together with the fact that a bag $\mathcal{B}_i$ is the union of $\log_2 \frac{1}{\varepsilon} + 2$ sets X_0^k. Lemma 13 allows to conclude. ◀

5 Capacitated Vehicle Routing

5.1 PTAS for Bounded Highway Dimension

The algorithm works as follows. The input graph G is embedded into a host graph H of bounded treewidth using the embedding given in Theorem 4. The algorithm then optimally solves the Capacitated Vehicle Routing problem with capacity Q for H, using a classical dynamic programming approach (described in the full version). The solution for H is then *lifted* to a solution in G: for each tour in the solution for H, a tour in G that visits the same clients in the same order is added to the solution for G.

We show that the embedding given in Theorem 4 is such that an optimal solution in the host graph H gives a $(1+\varepsilon)$ solution in G. Furthermore, the embedding ensures that H has small treewidth, allowing Capacitated Vehicle Routing to be solved exactly in polynomial time using dynamic programming. Putting these together gives Theorem 2.

Given an embedding with the properties described in Theorem 4, all that remains in proving Theorem 2 is showing how to solve Capacitated Vehicle Routing optimally on the host graph H and proving that such an optimal solution has a corresponding *near-optimal* solution in G. We do so in the following two lemmas (the first is proved in the full version of the paper)

▶ **Lemma 20.** *Given a graph with bounded treewidth ω and a capacity $Q > 0$,* Capacitated Vehicle Routing *can be solved optimally in $n^{O(\omega Q)}$ time.*

▶ **Lemma 21.** *For an embedding with the properties given by Theorem 4, the cost of an optimal solution in the host graph H is within a $(1+O(\varepsilon))$-factor of the cost of the optimal solution in the guest graph G.*

Proof. Let OPT_H be the optimal solution in the host graph H and OPT_G be the optimal solution in G. A solution is described by the order in which the clients and the depot are visited: $(u, v) \in S$ indicates that the solution S visits the client v immediately after visiting u. We want to prove that $\text{cost}_G(\text{OPT}_H) \leq (1 + O(\varepsilon))\text{cost}_G(\text{OPT}_G)$.

First, since $d_G \leq d_H$, $\text{cost}_G \leq \text{cost}_H$. Second, the solution OPT_G is also a solution in the host graph H, since the vertices of G and H are the same. So, by definition of OPT_H, $\text{cost}_H(\text{OPT}_H) \leq \text{cost}_H(\text{OPT}_G)$. It is therefore sufficient to prove that $\text{cost}_H(\text{OPT}_G) \leq (1 + O(\varepsilon))\text{cost}_G(\text{OPT}_G)$.

By definition of cost, $\mathrm{cost}_H(\mathrm{OPT}_G) = \sum_{(u,v)\in\mathrm{OPT}_G} d_H(u,v)$. Applying Theorem 4 gives

$$\mathrm{cost}_H(\mathrm{OPT}_G) \leq \sum_{(u,v)\in\mathrm{OPT}_G} d_G(u,v) + O(\varepsilon)(d_G(s,u) + d_G(s,v))$$

The right side of the inequality can be rewritten as

$$\underbrace{\sum_{(u,v)\in\mathrm{OPT}_G} d_G(u,v)}_{=\ \mathrm{cost}_G(\mathrm{OPT}_G)} \quad + \quad \underbrace{O(\varepsilon) \sum_{(u,v)\in\mathrm{OPT}_G} d_G(s,u) + d_G(s,v)}_{=\ O(\varepsilon)\sum_{v\in Z} 2d_G(s,v)\ \leq\ O(\varepsilon)Q\mathrm{cost}_G(\mathrm{OPT}_G)\quad (*)}$$

To get the inequalities $(*)$, it is enough to remark that OPT_G visits every client exactly once and then to apply Lemma 6. As Q is constant, the whole inequality becomes

$$\mathrm{cost}_H(\mathrm{OPT}_G) \leq \mathrm{cost}_G(\mathrm{OPT}_G) + O(\varepsilon)\mathrm{cost}_G(\mathrm{OPT}_G) = (1+O(\varepsilon))\mathrm{cost}_G(\mathrm{OPT}_G) \qquad ◀$$

References

1 Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. VC-dimension and shortest path algorithms. In *International Colloquium on Automata, Languages, and Programming*, pages 690–699. Springer, 2011.

2 Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. Highway dimension and provably efficient shortest path algorithms. *Journal of the ACM*, 63(5):41:1–41:26, 2016.

3 Ittai Abraham, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 782–793. SIAM, 2010.

4 Anna Adamaszek, Artur Czumaj, and Andrzej Lingas. PTAS for k-tour cover problem on the plane for moderately large values of k. *International Journal of Foundations of Computer Science*, 21(6):893–904, 2010. `doi:10.1142/S0129054110007623`.

5 Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for Euclidean k-medians and related problems. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 106–113, 1998.

6 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

7 Tetsuo Asano, Naoki Katoh, and Kazuhiro Kawashima. A new approximation algorithm for the capacitated vehicle routing problem on a tree. *Journal of Combinatorial Optimization*, 5(2):213–231, 2001.

8 Tetsuo Asano, Naoki Katoh, Hisao Tamaki, and Takeshi Tokuyama. Covering points in the plane by k-tours: a polynomial approximation scheme for fixed k. *IBM Tokyo Research Laboratory Research Report RT0162*, 1996.

9 Tetsuo Asano, Naoki Katoh, Hisao Tamaki, and Takeshi Tokuyama. Covering points in the plane by k-tours: towards a polynomial time approximation scheme for general k. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 275–283, 1997.

10 M. Bădoiu, S. Har-Peled, and P. Indyk. Approximate clustering via core-sets. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 250–257, 2002.

11 H. Bast, Stefan Funke, and Domagoj Matijevic. Ultrafast shortest-path queries via transit nodes. In Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors, *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 175–192. American Mathematical Society, 2009.

12 H. Bast, Stefan Funke, Domagoj Matijevic, Peter Sanders, and Dominik Schultes. In transit to constant time shortest-path queries in road networks. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 46–59. SIAM, 2007.

13 Amariah Becker, Philip N. Klein, and David Saulpic. Polynomial-time approximation schemes for k-center and bounded-capacity vehicle routing in metrics with bounded highway dimension. *arXiv:1707.08270*, 2017.

14 Amariah Becker, Philip N Klein, and David Saulpic. A quasi-polynomial-time approximation scheme for vehicle routing on planar and bounded-genus graphs. In *25th Annual European Symposium on Algorithms (ESA)*, volume 87, pages 12:1–12:15, 2017.

15 Jaroslaw Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Transactions on Algorithms*, 13(2), 2017.

16 Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local search yields approximation schemes for k-means and k-median in Euclidean and minor-free metrics. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2016.

17 Aparna Das and Claire Mathieu. A quasipolynomial-time approximation scheme for Euclidean capacitated vehicle routing. *Algorithmica*, 73(1):115–142, 2015.

18 Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC)*, 1988.

19 Andreas Emil Feldmann. Fixed parameter approximations for k-center problems in low highway dimension graphs. In *Proceedings, Part II, of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 588–600. Springer-Verlag, 2015.

20 Andreas Emil Feldmann, Wai Shing Fung, Jochen Könemann, and Ian Post. A $(1+\varepsilon)$-embedding of low highway dimension graphs into bounded treewidth graphs. In *Proceedings, Part I, of the International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 469–480. Springer, 2015.

21 Bruce L Golden and Richard T Wong. Capacitated arc routing problems. *Networks*, 11(3):305–315, 1981.

22 Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

23 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31(1):228–248, 1999.

24 Anupam Gupta, Robert Krauthgamer, and James R Lee. Bounded geometries, fractals, and low-distortion embeddings. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 534–543. IEEE, 2003.

25 Venkatesan Guruswami and Piotr Indyk. Embeddings and non-approximability of geometric problems. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2003.

26 Mark Haimovich and A. H. G. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10(4):527–542, 1985.

27 Shin Hamaguchi and Naoki Katoh. A capacitated vehicle routing problem on a tree. In *International Symposium on Algorithms and Computation*, pages 399–407. Springer, 1998.

28 Dorit S Hochbaum and David B Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

29 Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

30 Michael Khachay and Roman Dubinin. PTAS for the Euclidean capacitated vehicle routing problem in $\mathbb{R}^d$. In *Proceedings of the 9th International Conference on Discrete Optimization and Operations Research (DOOR)*, pages 193–205. Springer, 2016.

31 Ján Plesník. On the computational complexity of centers located in a graph. *Aplikace matematiky*, 25(6):445–452, 1980.

32 David B Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 265–274. ACM, 1997.

Fine-grained Lower Bounds on Cops and Robbers

Sebastian Brandt
ETH Zürich
Zürich, Switzerland
brandts@ethz.ch

Seth Pettie[1]
University of Michigan
Ann Arbor, MI, USA
pettie@umich.edu

Jara Uitto[2]
ETH Zürich
Zürich, Switzerland
jara.uitto@inf.ethz.ch

Abstract

Cops and Robbers is a classic pursuit-evasion game played between a group of g cops and one robber on an undirected N-vertex graph G. We prove that the complexity of deciding the winner in the game under optimal play requires $\Omega\left(N^{g-o(1)}\right)$ time on instances with $O(N\log^2 N)$ edges, conditioned on the Strong Exponential Time Hypothesis. Moreover, the problem of calculating the minimum number of cops needed to win the game is $2^{\Omega(\sqrt{N})}$, conditioned on the weaker Exponential Time Hypothesis. Our conditional lower bound comes very close to a conditional *upper bound*: if Meyniel's conjecture holds then the cop number can be decided in $2^{O(\sqrt{N}\log N)}$ time.

In recent years, the Strong Exponential Time Hypothesis has been used to obtain many lower bounds on classic combinatorial problems, such as graph diameter, LCS, EDIT-DISTANCE, and REGEXP matching. To our knowledge, these are the first conditional (S)ETH-hard lower bounds on a strategic game.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Cops and Robbers

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.9

1 Introduction

The game of *Cops and Robbers* is a two-player perfect information game played on a graph. One player is the *cop player*, who is identified with a set of g cops[3] occupying vertices of the graph. The other player is the *robber player*, who is identified with a single robber occupying some vertex of the graph. The game begins by the cop player placing the set of cops on the graph. Once she has decided the locations of the cops, it is the turn of the robber player to do the same.

Then, taking turns and initiated by the cop player, the players are allowed to move their pieces along the edges of the graph, where a turn of a player consists of moving all pieces the

[1] Supported by NSF grants CCF-1514383 and CCF-1637546.
[2] Partially supported by ERC Grant No. 336495 (ACDC).
[3] We decided to denote the number of cops by g as opposed to the "standard" k to avoid confusion later with the parameter k for k-CNF.

© Sebastian Brandt, Seth Pettie, and Jara Uitto;
licensed under Creative Commons License CC-BY

player is identified with to an adjacent vertex. We assume that the graph is reflexive, i.e., a player is allowed to let a piece stay in the vertex it is currently occupying. The goal of the cop player is to *capture* the robber, i.e., move at least one cop to the vertex occupied by the robber. Conversely, the goal of the robber is to avoid being captured indefinitely. We say that a graph G is *g-cop-win* if there is a strategy for g cops to guarantee capture of the robber. Furthermore, we call the smallest integer g such that G is a g-cop-win graph the *cop number* of G and denote it by $c(G)$. Notice that any graph with n vertices is n-cop-win.

In this paper, we study the computational complexity of determining the cop number of a given input graph. It is known from previous work by Berarducci and Intrigila [5] that, for a *fixed* g, one can check in polynomial time whether $c(G) \leq g$. On the other hand, it was recently shown by Kinnersley that, for a *non-fixed* g, i.e., that can be a function of n, deciding whether $c(G) \leq g$ is EXPTIME-complete [22].

Perhaps the most famous and intriguing problem in the field of cops and robbers is Meyniel's conjecture, that states that $O(\sqrt{n})$ cops *always* suffice to capture the robber in any n-vertex graph [17]. Towards proving this conjecture, it is known that there exist graphs with cop number $\Theta(\sqrt{n})$ [26], and that $n/2^{(1+o(1))\sqrt{\log n}}$ cops always suffice to capture the robber; see Scott and Sudakov [30], or Lu and Peng [24] for a similar bound. Combining this upper bound with an $n^{O(g)}$ algorithm for checking whether the cop number is at most g [5], the cop number can always be computed in $n^{n/2^{(1+o(1))\sqrt{\log n}}}$ time. Moreover, assuming that Meyniel's conjecture is true, this upper bound reduces to $n^{O(\sqrt{n})}$. Hence, under this assumption $2^{\Omega(\sqrt{n}\log n)}$ is the best lower bound that we can hope to achieve.

But how close to this bound is it possible to get? While the result by Kinnersley shows EXPTIME-completeness, it gives relatively loose guarantees on the actual value in the exponent of the runtime. Since the completeness proof goes through a series of reductions [22, 31] and the size of the input graph grows (polynomially) in these reductions, the lower bound by Kinnersley "only" gives a $2^{n^{1/5}}$ lower bound.[4]

Our work can be seen as a step towards finding the right asymptotic bound in the exponent. Furthermore, our construction is quite simple and, in particular, gives rise to very concise and easy to understand strategies for the players. To state our main results, we recall the satisfiability problem and the definitions of the exponential time hypotheses below.

▶ **Definition 1.** Let c_k be the smallest value such that instances of k-CNF-SAT with m clauses and n variables can be solved in $2^{(c_k+o(1))n}\,\mathrm{poly}(m)$ time. The *Exponential Time Hypothesis* (ETH) is that $c_k > 0$ for all $k \geq 3$. The *Strong Exponential Time Hypothesis* is that $\lim_{k\to\infty} c_k = 1$, i.e., k-CNF-SAT requires $2^{(1-o(1))n}$ time for any non-constant $k = k(n)$.

Conditioning on the Exponential Time Hypothesis and the Strong Exponential Time Hypothesis, we prove the following theorems. We want to emphasize that Theorem 2 is optimal up to a constant factor in the exponent and Theorem 3 and Theorem 4 are optimal up to a $\log N$ factor in the exponent, in the case of Theorem 3 under the assumption of Meyniel's conjecture. Furthermore, a potentially interesting detail of Theorem 2 is that it works for *any* $g \geq 2$, i.e., not only when g grows large.

[4] Suppose an ABF game [31] is played on a CNF formula with ℓ variables and $O(\ell)$ clauses. Kinnersley [22] reduces this to a *lazy cops and robbers with protection* game on $O(\ell^2)$ vertices, $O(\ell^3)$ edges, and $\ell + O(1)$ cops. Given any such game with n vertices, m edges, and g cops, Kinnersley [22] reduces it to an equivalent cops and robbers with protection game on $O(gn+m)$ vertices, $O(n(g^2+m))$ edges, and g cops. Mamino's reduction [25] from cops and robbers with protection to standard cops and robbers transforms a game with parameters n, m, g to $O(g^2 n), O(g^4 m), g$. Composing all three reductions, we arrive at a standard cops and robbers instance with $N = O(\ell^5)$ vertices, $O(\ell^9)$ edges, and $\ell + O(1)$ cops. If we need $2^{\Omega(\ell)}$ time to decide the winner of the original ABF game, then this gives, at the best, a $2^{\Omega(N^{1/5})}$ lower bound on deciding the cop number of an N-vertex graph.

▶ **Theorem 2.** *Fix an integer $g \geq 2$ and any $\delta > 0$. Conditioned on the Strong Exponential Time Hypothesis, the problem of deciding whether an N-vertex, M-edge graph has cop number at most g cannot be solved in $O(M^{g-\delta})$ time.*

In an informal sense, Theorem 2 can be interpreted as the statement that exploring almost all of the $O(M^{g+1})$, resp. $O(M^{g+2})$, possible game configurations and transitions between these configurations in a cops and robbers game with g, resp. $g + 1$, cops is unavoidable in order to determine whether the cop number is at most g or at least $g + 1$.

▶ **Theorem 3.** *Conditioned on the Exponential Time Hypothesis, the problem of calculating the cop number of an N-vertex graph cannot be solved in $2^{o(\sqrt{N})}$ time.*

As mentioned above, if Meyniel's conjecture is true, the lower bound given in Theorem 3 cannot be improved by more than a log-factor in the exponent. However, if Meyniel's conjecture turns out to be false and there is an infinite graph family requiring $\Omega(X(N))$ cops to capture the robber, for some function $X(N) = \omega(\sqrt{N})$, there is well-founded hope that our approach can be used to show that the problem of calculating the cop number of an N-vertex cannot be solved in $2^{o(X(n))}$, thereby staying in the realm of being only a log-factor away from the optimum. The reason for this hope is that the graphs we construct in order to infer our lower bound contain components that are essentially the hard instances for the $\Omega(\sqrt{N})$ lower bound on the cop number. Of course, it cannot be taken for granted that all proof details still work out if we replace these components with the hard instances for a larger lower bound on the cop number, but the simplicity of our construction suggests that this might indeed be the case.

▶ **Theorem 4.** *Let $g : \mathbb{N} \to \mathbb{R}$ be any function such that $g(x) = o(\sqrt{x})$ and $g(x+1) \leq g(x)+1$ for all positive integers x. Conditioned on the Exponential Time Hypothesis, the problem of deciding whether the cop number of an N-vertex graph is at most $g(N)$ cannot be solved in $2^{o(g(N))}$ time.*

Informally, Theorem 4 states that also for all ("natural") functions between constant functions and $\Theta(\sqrt{N})$, deciding whether the cop number of a graph is bounded by the function takes time exponential in the function. Similarly to the case of Theorem 3, in case Meyniel's conjecture turns out to be false, the range of functions for which Theorem 4 applies might be increased to include functions from $\omega(\sqrt{N})$ by adapting our graph construction in a straightforward way.

To the best of our knowledge, this is the first work to apply the (Strong) Exponential Time Hypothesis on a strategic game. In previous works, (S)ETH has been applied to, e.g., some well known combinatorial problems such as graph diameter [29], LCS [9, 10], EDIT-DISTANCE [3], and REGEXP matching [4, 11].

2 Related Work

The study of the game of Cops and Robbers was initiated by Quilliot [27] in 1978 and introduced independently a few years later by Nowakowski and Winkler [7]. Nowakowski and Winkler provided a full characterization of graphs where one cop can capture a robber which was later extended to the case of many cops by Clarke and MacGillivray [14]. One of the core questions related to the game is the *cop number* of a graph, which denotes the minimum number of cops required to capture the robber. A very early result by Aigner and Fromme states that 3 cops suffice to capture a robber on planar graphs and in the same work, they showed that any graph with girth at least 5 and minimum degree at least δ has a cop number of at least δ [2].

Later, Prałat showed that there are incidence graphs of projective planes that satisfy these properties for $\delta = \Omega(\sqrt{n})$ yielding the state-of-the-art lower bound for the cop number of any graph [26]. Given that Meyniel's [17] conjectured $O(\sqrt{n})$ upper bound holds, this bound is tight. The current best upper bound of $n/2^{(1+o(1))\sqrt{\log n}}$ [30] is far away from this though and improving it is perhaps the most crucial open problem in the field.

Beyond the existential question of determining the maximum cop number, there is the computational question. On the positive side, for fixed g, determining whether the cop number is at most g can be computed in polynomial time [5]. To the best of our knowledge, the current best algorithm runs in $O(n^{2g+3})$ time [6]. Many years later, this was contrasted by a negative result showing that for a non-fixed g, i.e., g can be a function of the number of vertices n, this question becomes NP-hard [16]. A bit later, it was shown by Mamino that this question is hard for PSPACE [25]. An interesting detail on this work is that it goes through a reduction to a variant called "Cops and Robbers with Protection". In this variant, edges are divided into protected and unprotected edges. The crux of the game is that the capture only occurs if a cop moves to the vertex occupied by the robber through an unprotected edge. In a recent breakthrough, Kinnersley managed to show that the stardard variant of the problem is actually EXPTIME-complete [22].

Even though the progress on the specific question of finding the cop number is fairly recent, other related questions in various graph classes have been studied long ago. For example, in the end of seventies and beginning of eighties, Adachi et al. studied a variant of the game where one cop is trying to prevent any of multiple robbers from reaching a "hole" in the graph [21, 1]. In their variant, the initial positions are fixed and the cop and exactly one robber have to move in each turn. They showed EXPTIME-completeness. For a survey of earlier complexity results, we refer to a survey by Johnson [20]

Goldstein and Reingold [18] studied a version of the game in which the cops and robbers have prescribed initial positions and the goal of the robber is to reach a specific vertex. They showed that in undirected graphs this variant of the game is EXPTIME-complete. In the same work, they showed that the directed version of the game, without fixing the initial positions, is also EXPTIME-complete.

For the curious reader, we point out that many of the results listed here are based on reductions to the ABF-problem that was shown to be EXPTIME-complete by Stockmeyer and Chandra [31]. Furthermore, for a great survey on the results of the game we refer the reader to the book by Bonato and Nowakowski [7].

3 Preliminaries

Let us give some definitions that are used throughout the paper.

▶ **Definition 5** (k-CNF-SAT)**.** The input to the k-CNF-SAT problem is a conjunction of one or more clauses, where each clause consists of a disjunction of at most k literals. The goal is to determine whether the formula is *satisfiable*, i.e., if there is a truth assignment of the variables such that the input formula evaluates to true.

Especially, we wish to specify what we mean by a *partial assignment* of variables in a logical formula consisting only of literals, disjunctions, and conjunctions. In a partial assignment, a subset of the variables is set to true/false and some may be left unassigned. A disjunctive clause $x_1 \vee x_2 \vee \dots \vee x_\ell$, for $\ell \geq 1$ is *satisfied* by a partial assignment if at least one literal in the clause has an assigned truth value and is true. We point out that this means that a disjunctive clause that contains both a variable and its negation can still be unsatisfied by a partial assignment. Throughout the paper, we denote the number of variables in a k-CNF-SAT instance by n, the number of vertices in a graph by N, the number of cops by g, and we reserve the letter k as the parameter for k-CNF-SAT.

4 The Construction

Fix a number $g \geq 2$ of cops and an integer $k \geq 3$. The technique we use to derive our main results is a reduction from k-CNF-SAT to the problem of deciding whether a graph is g-cop-win. To this end, we will start this section by describing how we transform any k-CNF formula with n variables and m clauses into an input graph for the Cops and Robbers game with g cops. Then we will prove that our graph construction has the property that the cops can win the game in the constructed graph iff the k-CNF formula is satisfiable. We will conclude the section by using this property to infer our lower bounds.

In the following we give an informal high-level overview of our construction. We say that vertex v *covers* a set of vertices S if v is adjacent to all vertices in S. Vertex v always covers itself. The constructed graph consists of two zones: one that is designed for the cops from which they can cover the whole graph if the k-CNF formula is satisfiable, and one for the robber in which he can evade capture indefinitely if the formula is unsatisfiable.

The cops' zone consists of $g2^{\lceil n/g \rceil}$ vertices, which represent certain partial assignments to groups of $\lceil n/g \rceil$ variables in the CNF formula. By occupying g non-conflicting partial assignments, the cops can collectively represent a total assignment to the variables. If this total assignment is satisfying, then it should cover *every* vertex in the robber area, leaving the robber nowhere to go. (Each vertex in the robber's zone is associated with a clause, which is covered by the cops if their collective assignment satisfies the clause.) On the other hand, if no satisfying assignment exists, then the robber must always be able to move to some vertex not covered by any cop.

If the cops and robbers agreed to stay in their own zones then the construction of the robber's zone could be very simple: m vertices (one for each clause) arranged in a clique suffices. Of course, both the robber and the cops are free to roam over the whole graph, so we need to add extra mechanisms to dissuade the robber from entering the cops' zone, and protect the robber against any cops entering the robber's zone. To protect the robber, we make the subgraph induced by the robber's zone a girth-6 graph,[5] which means that any cop that enters the robber's zone can never cover more than one neighbor of the robber, leaving many options for the robber to escape. The mechanism to dissuade the robber from entering the cops' zone is more subtle; it ensures that any robber that does this loses in two turns, regardless of whether the k-CNF formula is satisfiable or not.

Because we are interested in lower bounds *as a function of input size*, it is important to keep the graph as sparse as possible. Many transformations on cops and robbers games (e.g., [22, 23, 25]) create very dense graphs, sometimes having $\Omega(n^2)$ edges. Parts of our construction could be simplified by introducing large cliques, but this would weaken the resulting (conditional) lower bounds. This concludes the informal overview; in the following, we will give a formal description of our graph construction.

Let $\phi = C_1 \wedge \cdots \wedge C_m$ be a k-CNF formula over the variable set $\mathcal{V} = \{v_1, \ldots, v_n\}$, where the ith clause is $C_i = x_{i,1} \vee \cdots \vee x_{i,k}$ and each $x_{i,j}$ is a variable or its negation. The variable set is partitioned into $g \geq 2$ groups $\mathcal{V}_1, \ldots, \mathcal{V}_g$ of at most $\lceil n/g \rceil$ variables each. For reasons that will become clear later, it is desirable that the formula has the property that any *partial*

[5] "Girth" is the length of the shortest cycle.

$$\phi'\left\{\begin{array}{ll} \overset{C_1}{(v_1 \lor v_2 \lor v_3)} \land \overset{C_2}{(\neg v_3 \lor v_4)} \land \overset{C_3}{(v_2 \lor \neg v_4)} & \}\ \phi \\ \quad \land (v_1 \lor \neg v_1 \lor v_2 \lor \neg v_2) \land (v_3 \lor \neg v_3 \lor v_4 \lor \neg v_4) & \}\ C_4 \land C_5 \end{array}\right.$$

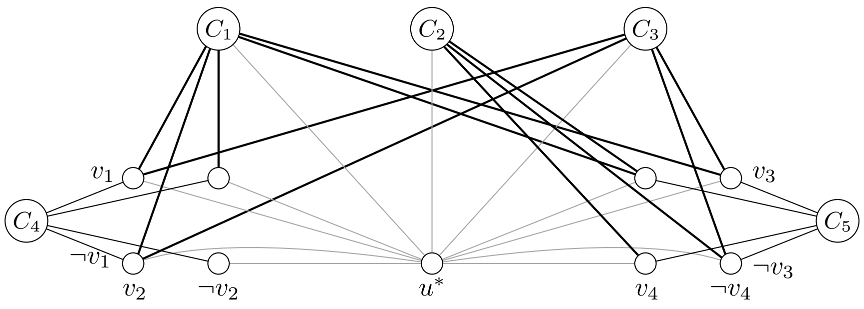


Figure 1 A schematic and simplified illustration of our graph construction in the case of two cops and a k-CNF formula ϕ with four variables and three clauses. Notice that vertex u^* is not connected to the two extra clauses C_4 and C_5. Each vertex in the figure labeled with C_i for some i corresponds to the set of vertices in B with clause-type i. The small vertices correspond to the partial truth assignments and are connected to the clause vertices that they satisfy (i.e., the corresponding literal is contained in these clauses). Notice that C_4 and C_5 are only covered by the cops if they occupy a set of vertices that corresponds to assigning a value to all variables. The edges between vertices in the C_i, i.e., between vertices in B, are not shown in Figure 1. For an illustration of these edges, see Figure 2.

satisfying truth assignment must set at least one variable in each group. To this end, we supplement ϕ with g extra clauses. Define ϕ' as follows.

$$\phi' = C_1 \land \cdots \land C_m \land C_{m+1} \land \cdots \land C_{m+g}$$
$$\text{where } C_{m+i} = \bigvee_{v \in \mathcal{V}_i} (v \lor \neg v)$$

Observe that ϕ' is satisfiable iff ϕ is since *any* total assignment to $\mathcal{V}$ automatically satisfies each of the clauses $C_{m+1}, \ldots, C_{m+g}$. Define $\overline{m} = m + g$.

The next step is to convert ϕ' to the graph G on which the Cops and Robbers game will be played. See Figure 1 for a simplified illustration of a graph constructed from a k-CNF-SAT instance.

Vertices

The vertex set $V(G)$ is $A_1 \cup \cdots \cup A_g \cup B \cup \{u^\star\}$, where there is a vertex $u \in A_i$ for each truth assignment $\psi_u : \mathcal{V}_i \to \{\mathsf{T}, \mathsf{F}\}$ to the ith variable group. The set B consists of $\Theta(\overline{m}^2)$ vertices, each of which is associated with one of the $\overline{m}$ clauses in ϕ'. If $u \in B$, clause$(u) \in [\overline{m}]$ indicates the clause index associated with u, and we say that u has *clause-type* clause(u). The role of $u^\star$ will be revealed shortly. In total, $|V(G)| = O(g2^{n/g} + \overline{m}^2)$.

Edges

The edge set $E(G)$ includes edges of three types:

Satisfaction Edges. Edges join partial assignments to clauses iff the partial assignment satisfies the clause:

$$\{\{u, u'\} \mid u \in A_i, u' \in B, \text{clause}(u') = q, \text{ and } \psi_u \text{ satisfies } C_q\} \subset E(G)$$

Special $u^\star$ Edges. In some ways, $u^\star$ functions like an assignment that magically satisfies all clauses C_i where $i \in [m]$, but none where $i \in [\overline{m}]\backslash[m]$. It is also adjacent to all vertices in $A_1, \dots, A_g$.

$$\{\{u^\star, u\} \mid \text{Either } u \in A_1 \cup \dots \cup A_g \text{ or } u \in B \text{ and clause}(u) \in [m]\} \subset E(G),$$

High Girth Subgraph. The subgraph of G induced by B has $\Theta(|B|^{3/2}) = \Theta(\overline{m}^3)$ edges and girth at least 6. Moreover, for each $u \in B$ and each $q \in [\overline{m}]$,

$$|\{u' \in B \mid \{u, u'\} \in E(G) \text{ and } \text{clause}(u') = q\}| \geq 1 \ .$$

I.e., each B-vertex has at least one neighbor of each clause-type.

It is not immediate from the description that the subgraph induced by B actually exists. We construct such a graph and clause-assignment now. Let p be the first prime greater than $\overline{m}$, so $p = \Theta(\overline{m})$, by Bertrand's postulate [13]. Define line(s,t) to be the line in $\mathbb{Z}_p^2$ with *slope* s and *offset* t:

$$\text{line}(s,t) = \{(i,j) \in \mathbb{Z}_p^2 \mid i \cdot s + t \equiv j \pmod p\}.$$

The set B consists of $2p^2$ vertices $\{w_{i,j}, l_{s,t} \mid (i,j),(s,t) \in \mathbb{Z}_p^2\}$, where $w_{i,j}$ represents the point (i,j) and $l_{s,t}$ represents line(s,t). The subgraph induced by B is simply the point-line incidence graph, i.e.,

$$\{w_{i,j}, l_{s,t}\} \in E(G) \Leftrightarrow (i,j) \in \text{line}(s,t) \ .$$

We restate some properties of this graph that were shown in previous work [8]. See [12, 15, 28, 33, 32, 26] for other constructions with essentially the same properties.

▶ **Lemma 6.** *Consider the p^2 points and p^2 lines indexed by $\mathbb{Z}_p^2$.*
1. *The intersection of two lines contains at most one point.*
2. *Two points are contained in at most one common line.*
3. *For any point (i,j) and any slope s, there exists some* line(s,t) *containing (i,j).*
4. *For any* line(s,t) *and index i, there is some point $(i,j) \in$* line(s,t).

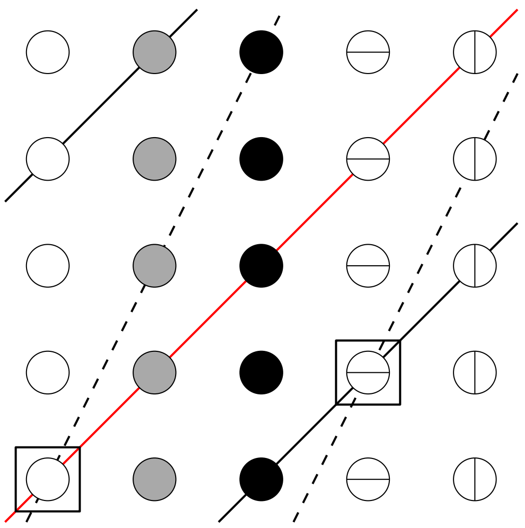
Properties (1) and (2) of Lemma 6 imply that the subgraph induced by B has no 4-cycles. Since it is clearly bipartite, it must have girth (at least) 6. We use properties (3) and (4) of Lemma 6 to design a good clause-assignment function clause : $B \to [\overline{m}]$. In particular,

$$\text{For points, } \text{clause}(w_{i,j}) = i + 1$$
$$\text{For lines, } \text{clause}(l_{s,t}) = s + 1$$

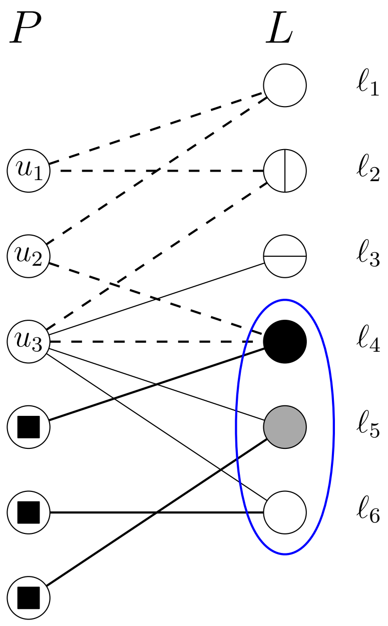
Since $p \geq \overline{m}$, it follows that for each clause index $q \in [\overline{m}]$, every point $w_{i,j}$ has at least one neighboring line with clause-type q, and every line $l_{s,t}$ has at least one neighboring point with clause-type q. See Figure 2 for an illustration.

This concludes the description of graph G. It is straightforward to construct G in time linear in the number of edges, which is $O(\overline{m}^2 g 2^{n/g} + \overline{m}^3)$. The following lemma shows that the construction indeed satisfies its purpose, i.e., the constructed graph G is g-cop-win if and only if the k-CNF formula ϕ is satisfiable.

▶ **Lemma 7.** *In the Cops and Robbers game on G with g cops, the cops have a winning strategy iff ϕ is satisfiable.*



(a) The subgraph B can be seen as a set of vertices/-points and lines on a plane. In the figure, the points associated with the same clause are illustrated by the same color. Lines with slope 1 and offsets 0 and 3 are illustrated by solid red and black lines, respectively. Line with slope 2 and offset 0 is illustrated by a dashed line. The two unique intersections of the non-parallel lines are emphasized with black boxes.



(b) Concretely, the subgraph B is a bipartite graph with points on one side (left) and the lines on the other. Since every two lines have at most one intersection point, at most one neighbor of a point vertex u_3 can be covered by any other point vertex (see black boxes in the figure). Hence, 5 cops are needed to cover the neighbors of u_3. The same line of reasoning holds for any line vertex.

Figure 2 The subgraph depicted as a set of points and lines on a plane and as a bipartite graph. Notice that any cycle starting from a point vertex must pass through at least 3 line vertices. Therefore, the girth of the graph is at least 6. This is illustrated by the dashed edges incident on vertices u_1, u_2, and u_3 on the right. Notice that the pictures above are not inferred from each other.

Proof. Suppose $\psi : \mathcal{V} \to \{\mathsf{T}, \mathsf{F}\}$ is a satisfying total assignment, decomposed into partial assignments $\psi_{u_1}, \dots, \psi_{u_g}$, where ψ_{u_i} is associated with $u_i \in A_i$. In their initial move, the cops position themselves on $u_1, \dots, u_g$. At this point they cover all vertices in $B \cup \{u^\star\}$, but leave the remaining vertices in $A_1 \cup \dots \cup A_g$ uncovered. Without loss of generality we can assume that the robber begins at a vertex in $A_1 \backslash \{u_1\}$. In the next move, the cops stay put, except for the cop on u_2, which moves to $u^\star$. At this point all B-vertices with clause-types in $[m]$ are covered by the cop on $u^\star$, and those with clause-type $m + 1$ are covered by the cop on u_1. The robber, being in A_1, can move once more or stay put, but is immediately caught by the cop on u_1 or $u^\star$ in the next turn.

Now consider the case where ϕ is unsatisfiable. We show that the robber has a winning strategy such that it never leaves the set B. Consider any moment in the middle of the game, after the cops have moved to vertices $w_1, \dots, w_g$. The robber is located at some $w' \in B$ and may be forced to move if w' is in the neighborhood of $w_1, \dots, w_g$. Let $z \geq 0$ be the number of cops that are located at some vertex in B. First consider the case that at least $z + 1$ of the A_i do not contain any cop, and without loss of generality, let $A_1, \dots, A_{z+1}$ contain no cop. By the properties of G, the robber is adjacent to a set of vertices $S = \{w'_1, \dots, w'_{z+1}\} \subset B$, where clause$(w'_i) = m + i$. None of the S-vertices are covered by the $g - z$ cops stationed in $A_{z+2} \cup \dots \cup A_g \cup \{u^\star\}$. Since the subgraph induced by B has girth at least 6, each of the remaining z cops can cover at most one S-vertex, hence at least one S-vertex is not covered by any cop, and the robber can move there without being captured. Now consider the other case, i.e., that exactly z of the A_i, say $A_1, \dots, A_z$, do not contain any cop. Then the $g - z$ sets $A_{z+1}, \dots, A_g$ contain exactly one cop each. Assume without loss of generality that $w_{z+1} \in A_{z+1}, \dots, w_g \in A_g$, and let ψ' be the partial assignment obtained by combining the partial assignments $\psi_{w_{z+1}}, \dots, \psi_{w_g}$. Since ϕ is unsatisfiable, there is a clause from ϕ not satisfied by ψ', say clause C_q. Similarly to the previous case, by the properties of G, the robber is adjacent to a set of $z+1$ vertices $S = \{w'_1, \dots, w'_z, w'\} \subset B$, where clause$(w'_i) = m + i$ and clause$(w') = q$. Again, none of the S-vertices is covered by the $g - z$ cops stationed in $A_{z+1} \cup \dots \cup A_g$ and the remaining z cops can cover at most z S-vertices. Now, with the same argumentation as in the previous case, it follows that there is an S-vertex the robber can move to without being captured.

The arguments above apply to any stage in the middle of the game; the same arguments show that if ϕ is unsatisfiable, the robber has a safe *first* move, after the cops choose their initial positions. ◀

We also obtain the following curious observation from our construction. Later, we use the observation to slightly strengthen our results, but we also believe that it is a property of the construction that is of independent interest.

▶ **Observation 8.** *Recall the vertex set of G is $V(G) = A_1 \cup \dots \cup A_g \cup B \cup \{u^\star\}$. Then the cop number of G is either g or $g+1$.*

Proof. If there are $g+1$ cops, they can position themselves on vertices $u_1, \dots, u_g, u^\star$ with $u_i \in A_i$. Then, since $u^\star$ is connected to all vertices in $A_1 \cup \dots \cup A_g \cup B$ except those in B with clause-type in $[\overline{m}] \backslash [m]$, and for each i, the cop in u_i covers all vertices in B with clause-type $m + i$, the cops cover the entire graph and hence can capture the robber in the following turn. If, on the other hand, there are at most $g - 1$ cops, then the robber has a simple winning strategy by always moving to a vertex in B with clause-type in $[\overline{m}] \backslash [m]$ that is not covered by any cop. By analogous arguments to the ones used in the proof of Lemma 7, such a vertex always exists. ◀

5 Hardness of Finding the Cop Number

Quickly before going into the proofs of our main theorems, we point out a small technical detail. The input k-CNF-SAT instance that we reduce to the Cops and Robbers instance may contain a very large number of clauses. This would then imply that our graph constuction has many edges, up to around $\hat{m}^4$ edges, where $\hat{m}$ is the number of input clauses. This would in turn result in a running time for our construction that is too large for our purposes. We can work around this problem by using the sparsification lemma [19], which, for any chosen $\epsilon > 0$, reduces an arbitrary k-CNF-SAT instance to $2^{\epsilon n}$ k-CNF-SAT instances with at most $c(k, \epsilon) \cdot n$ clauses each, where $c(k, \epsilon)$ is a function independent of n.

Next, we prove Theorem 2, i.e., that under the Strong Exponential Time Hypothesis, the time needed to decide whether the cop number is at most some fixed g grows exponentially as a function of g. A proof sketch goes as follows. We are given a k-CNF-SAT instance with n variables and $O(n)$ clauses. We obtain a graph with roughly $2^{n/g}$ vertices and edges from our construction. Being able to solve our cop number decision problem in $M^{g-\delta} = \left(2^{n/g}\right)^{g-\delta} \ll 2^n$ time yields a contradiction to the Strong Exponential Time Hypothesis.

▶ **Theorem 2.** *Fix an integer $g \geq 2$ and any $\delta > 0$. Conditioned on the Strong Exponential Time Hypothesis, the problem of deciding whether an N-vertex, M-edge graph has cop number at most g cannot be solved in $O(M^{g-\delta})$ time.*

Proof. Let $\hat{\phi}$ be an instance of k-CNF-SAT with $\hat{m}$ clauses and n variables, and let $\epsilon > 0$. Using the sparsification lemma, in $\text{poly}(n) \cdot 2^{\epsilon n}$ time we can reduce $\hat{\phi}$ to $2^{\epsilon n}$ instances of k-CNF-SAT, each having at most $m = c(k, \epsilon) \cdot n$ clauses. Let ϕ be one of those instances, and let G be the graph obtained by applying our graph construction to ϕ. G is an N-vertex, M-edge graph, where $N = \Theta(g 2^{n/g} + m^2) = \Theta(2^{n/g})$ and $M = O(m^2 \cdot N) = O(N \log^2 N)$. Thus, if we can decide in $O(M^{g-\delta}) = O(\text{poly}(m) N^{g-\delta})$ time whether G has cop number g, we can determine the satisfiability of $\hat{\phi}$ in $\text{poly}(m) 2^{\epsilon n} \cdot N^{g-\delta} = \text{poly}(n) 2^{n(\epsilon+1-\delta/g)}$ time, by Lemma 7. The calculations above do not depend on the value of k, so setting $\epsilon < \delta/g$ contradicts the Strong Exponential Time Hypothesis. ◀

Next, we provide the proof for Theorem 3. We note that this result can also be obtained from extending the proof of Theorem 4 to functions $g(x) = \Theta(x)$ (which requires some extra care), but this special case is much cleaner to prove and has all the same ingredients. The main difference is in the simplicity of calculations. The difference to the proof of Theorem 2 is that since g is a function of n, we can set $g = n$ and the graph becomes much smaller in terms of the number of variables of the input k-CNF formula. As a consequence, the dominating part of the constructed graph G w.r.t size is now B (and not the A_i, as in the proof of Theorem 2).

▶ **Theorem 3.** *Conditioned on the Exponential Time Hypothesis, the problem of calculating the cop number of an N-vertex graph cannot be solved in $2^{o(\sqrt{N})}$ time.*

Proof. Fix an arbitrarily small constant ϵ and an integer $k \geq 3$. Let $\hat{\phi}$ be an instance of k-CNF-SAT with $\hat{m}$ clauses and n variables, and ϕ be one of the $2^{\epsilon n}$ instances with $m = c(k, \epsilon) n$ clauses generated from the sparsification lemma. Use our graph construction to create a graph G from ϕ for a Cops and Robbers game with $g = n$ cops. G has $N = \Theta(g 2^{n/g} + (n+m)^2) = \Theta(m^2)$ vertices and $O(m^2 g 2^{n/g} + m^3) = O(m^3)$ edges. If we can determine the cop number of G in $2^{o(\sqrt{N})} = 2^{o(m)} = 2^{o(n)}$ time, we can determine the satisfiability of $\hat{\phi}$ in $\text{poly}(n) 2^{\epsilon n} \cdot 2^{o(n)} = \text{poly}(n) 2^{(\epsilon + o(1))n}$ time, by Lemma 7. Since ϵ can be made arbitrarily small, this contradicts the Exponential Time Hypothesis. ◀

As our last technical contribution, we show that one can replace the $\sqrt{N}$ in the exponent in Theorem 3 with essentially any reasonable function in N that is asymptotically smaller than $\sqrt{N}$ and obtain a lower bound for deciding whether the cop number of an input graph is bounded by this function. Basically, the statement of this theorem, combined with Observation 8 is that even when we know that the cop number is either $g(N)$ or $g(N) + 1$, the decision problem is hard.

▶ **Theorem 4.** *Let $g : \mathbb{N} \to \mathbb{R}$ be any function such that $g(x) = o(\sqrt{x})$ and $g(x+1) \leq g(x) + 1$ for all positive integers x. Conditioned on the Exponential Time Hypothesis, the problem of deciding whether the cop number of an N-vertex graph is at most $g(N)$ cannot be solved in $2^{o(g(N))}$ time.*

Proof. Let ϵ, k, $\hat{\phi}$, $\hat{m}$, n, ϕ and m be as in the proof of Theorem 3. Use our graph construction to create a graph G from ϕ for a Cops and Robbers game with n cops and denote the number of vertices of G by N. Check whether $g(N) < n + 1$. Observe that since $g(x) = o(\sqrt{x})$ and $N = O(n^2)$, there is some constant n_0 such that $g(N) < n + 1$ for all possible k-CNF formulae ϕ with $n \geq n_0$ variables. Hence, if $g(N) \geq n + 1$, n is constant and we can decide in constant time whether ϕ is satisfiable.

Now consider the other case, i.e., $g(N) < n+1$. As we want to use the Exponential Time Hypothesis in order to infer a conditional lower bound on the time it takes to determine whether the cop number of an N-vertex graph is at most $g(N)$, we would like the constructed graph G to have the property that ϕ is satisfiable iff G has cop number at most $g(N)$. With the current construction of G we only have a similar property, namely, that ϕ is satisfiable iff G has cop number at most n, due to Lemma 7. But since $g(N) < n+1$, we can change G slightly, adding more and more vertices to G in a way that does not change the cop number of G, and in the end obtain a graph G' with the desired property. More specifically, we obtain G' from G by appending a path of r vertices to some arbitrarily chosen vertex u of G, where r is the smallest non-negative integer such that $g(N+r) \geq n$.

Set $N' = N + r$. Due to the properties of our function g, we know that $g(N') < n+1$. Note that the cop number of G' is the same as the cop number of G: In the case that ϕ is unsatisfiable, our robber strategy still works with the same arguments as in G. In the case that ϕ is satisfiable, the cops can simply perform the same strategy as in G, where they assume that the robber is in u if the robber is actually in one of the newly appended vertices. With this strategy, after 2 turns per player, the cops have captured the robber or at least one cop ends up at u while the robber is in one of the new vertices, in which case the robber can be captured by letting the cop traverse the appended path to the other end.

From the discussion above it follows that ϕ is satisfiable iff G' has cop number at most $g(N')$. Hence, if the problem of deciding whether the cop number of an N-vertex graph is at most $g(N)$ can be solved in $2^{o(g(N))} = 2^{o(n)}$ time, we can determine the satisfiability of $\hat{\phi}$ in $\text{poly}(n)2^{\epsilon n} \cdot 2^{o(n)} = \text{poly}(n)2^{(\epsilon + o(1))n}$ time. Since ϵ can be made arbitrarily small, this contradicts the Exponential Time Hypothesis. ◀

References

1 Akeo Adachi, Shigeki Iwata, and Takumi Kasai. Some combinatorial game problems require $\omega(n^k)$ time. *J. ACM*, 31(2):361–376, 1984.

2 Martin Aigner and Michael Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8:1–12, 1984.

3 A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 51–58, 2015.

4 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 457–466, 2016. `doi:10.1109/FOCS.2016.56`.

5 A. Berarducci and B. Intrigila. On the cop number of a graph. *Advances in Applied Mathematics*, 14(4):389–403, 1993.

6 Anthony Bonato and Ehsan Chiniforooshan. Pursuit and evasion from a distance: Algorithms and bounds. In *Proceedings of the Meeting on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 1–10, 2009.

7 Anthony Bonato and Richard J. Nowakowski. *The Game of Cops and Robbers on Graphs*, volume 61. American Mathematical Soc., 2011.

8 Sebastian Brandt, Yuval Emek, Jara Uitto, and Roger Wattenhofer. A tight lower bound for the capture time of the cops and robbers game. In *44th International Colloquium on Automata, Languages, and Programming (ICALP), Warsaw, Poland*, 2017.

9 K. Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proceedings 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 661–670, 2014.

10 K. Bringmann and M. Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proceedings 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 79–97, 2015.

11 Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 307–318, 2017. doi:10.1109/FOCS.2017.36.

12 W. G. Brown. On graphs that do not contain a Thomsen graph. *Canad. Math. Bull.*, 9:281–285, 1966.

13 P. L. Chebyshev. Memoire sur les nombres premiers. *J. Math. Pures Appl.*, 17:366–390, 1852.

14 Nancy E. Clarke and Gary MacGillivray. Characterizations of k-copwin graphs. *Discrete Mathematics*, 312(8):1421–1425, 2012. doi:10.1016/j.disc.2012.01.002.

15 P. Erdős, A. Rényi, and V. T. Sós. On a problem of graph theory. *Studia Sci. Math. Hungar.*, 1:215–235, 1966.

16 Fedor V. Fomin, Petr A. Golovach, Jan Kratochvíl, Nicolas Nisse, and Karol Suchan. Pursuing a fast robber on a graph. *Theoretical Computer Science*, 411(7):1167–1181, 2010.

17 Peter Frankl. Cops and robbers in graphs with large girth and Cayley graphs. *Discrete Applied Mathematics*, 17(3):301–305, 1987. doi:10.1016/0166-218X(87)90033-3.

18 Arthur S. Goldstein and Edward M. Reingold. The complexity of pursuit on a graph. *Theor. Comput. Sci.*, 143(1), 1995.

19 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

20 David S. Johnson. The NP-completeness column: An ongoing guide. *Journal of Algorithms*, 4(4):397–411, 1983.

21 Takumi Kasai, Akeo Adachi, and Shigeki Iwata. Classes of pebble games and complete problems. *SIAM Journal on Computing*, 8(4):574–586, 1979.

22 William B. Kinnersley. Cops and robbers is EXPTIME-complete. *Journal of Combinatorial Theory, Series B*, 111:201–220, 2015.

23 William B. Kinnersley. Bounds on the length of a game of cops and robbers. *CoRR*, abs/1706.08379, 2017. arXiv:1706.08379.

24 Linyuan Lu and Xing Peng. On Meyniel's conjecture of the cop number. *Journal of Graph Theory*, 71(2):192–205, 2012.

25 Marcello Mamino. On the computational complexity of a game of cops and robbers. *Theoretical Computer Science*, 477:48–56, 2013.

26 Paweł Prałat. When does a random graph have constant cop number? *Australasian Journal of Combinatorics*, 46:285–296, 2010.

27 Alain Quilliot. *Jeux et Pointes Fixes sur les Graphes.* PhD thesis, Université de Paris VI, 1978.

28 I. Reiman. Über ein Problem von K. Zarankiewicz. *Acta. Math. Acad. Sci. Hungary*, 9:269–273, 1958.

29 L. Roditty and V. Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings 45th ACM Symposium on Theory of Computing (STOC)*, pages 515–524, 2013.

30 Alex Scott and Benny Sudakov. A bound for the cops and robbers problem. *SIAM Journal on Discrete Mathematics*, 25(3):1438–1442, 2011. doi:10.1137/100812963.

31 Larry J. Stockmeyer and Ashok K. Chandra. Provably difficult combinatorial games. *SIAM Journal on Computing*, 8(2):151–174, 1979.

32 J. Tits. Sur la trialité et certains groupes qui s'en déduisent. *Publ. Math. I.H.E.S.*, 2:14–20, 1959.

33 R. Wenger. Extremal graphs with no C^4s, C^6s, or C^{10}s. *J. Comb. Theory Ser. B*, 52(1):113–116, 1991.

A Polynomial Kernel for Diamond-Free Editing

Yixin Cao
Department of Computing, Hong Kong Polytechnic University, Hong Kong, China
yixin.cao@polyu.edu.hk
https://orcid.org/0000-0002-6927-438X

Ashutosh Rai
Department of Computing, Hong Kong Polytechnic University, Hong Kong, China
ashutosh.rai@polyu.edu.hk

R. B. Sandeep
Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Hungary; and Indian Institute of Technology Dharwad, Dharwad, India
sandeeprb@iitdh.ac.in

Junjie Ye
Department of Computing, Hong Kong Polytechnic University, Hong Kong, China
junjie.ye@polyu.edu.hk
https://orcid.org/0000-0003-3924-008X

Abstract

Given a fixed graph H, the H-free editing problem asks whether we can edit at most k edges to make a graph contain no induced copy of H. We obtain a polynomial kernel for this problem when H is a diamond. The incompressibility dichotomy for H being a 3-connected graph and the classical complexity dichotomy suggest that except for H being a complete/empty graph, H-free editing problems admit polynomial kernels only for a few small graphs H. Therefore, we believe that our result is an essential step toward a complete dichotomy on the compressibility of H-free editing. Additionally, we give a cubic-vertex kernel for the diamond-free edge deletion problem, which is far simpler than the previous kernel of the same size for the problem.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Kernelization, Diamond-free, H-free editing, Graph modification problem

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.10

Funding Supported in part by NSFC under grant 61572414, RGC under grants 152261 and 252026, and ERC under grant 725978.

1 Introduction

A graph modification problem asks whether one can apply at most k modifications to a graph to make it satisfy certain properties. By modifications we usually mean additions and/or deletions, and they can be applied to vertices or edges. Although other modifications are also considered, most results in literature are on vertex deletion and the following three edge modifications: edge deletion, edge completion, and edge editing (deletion/completion).

As usual, we use n to denote the number of vertices of the input graph. For each graph modification problem, one may ask three questions: (1) Is it NP-complete? (2) Can it be solved in time $f(k) \cdot n^{O(1)}$ for some function f, and if yes, what is the (asymptotically) best f? (3) Does it have a polynomial kernel? The first question concerns classic complexity, while the other two are about parameterized complexity [9, 6]. With parameter k, a problem is

© Yixin Cao, Ashutosh Rai, R. B. Sandeep, and Junjie Ye;
licensed under Creative Commons License CC-BY

fixed-parameter tractable (FPT) if it can be solved in time $f(k) \cdot n^{O(1)}$ for some function f. On the other hand, given an instance (G, k), a *kernelization algorithm* produces in polynomial time an equivalent instance (G', k') – (G, k) is a yes-instance if and only if (G', k') is a yes-instance – such that $k' \leq k$. It is a *polynomial kernel* if the size of G' is bounded from above by a polynomial function of k.

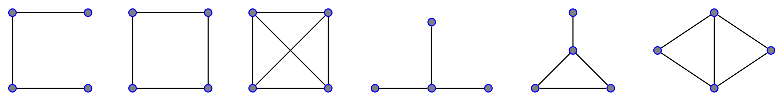
For hereditary properties, a classic result of Lewis and Yannakakis [13] states that all the vertex deletion problems are either NP-hard or trivial. In contrast, the picture for edge modification problems is far murkier. Earlier efforts for edge deletion problems [15, 7], though having produced fruitful concrete results, shed little light on a systematic answer, and it was noted that such a generalization is difficult to obtain.

A basic and ostensibly simple case of graph modification problems is to make the graph H-free, where H is a fixed graph on at least two vertices. (We say that a graph is H-free if it does not contain H as an induced subgraph.) For this special case, all the three questions have been satisfactorily answered for vertex deletion problems, at least in the asymptotic sense. All of them are NP-complete and FPT– indeed, H-free vertex deletion problems admit simple $|V(H)|^k \cdot n^{O(1)}$-time algorithms [2]. On the other hand, the reduction of Lewis and Yannakakis [13] excludes subexponential-time algorithms ($2^{o(k)} \cdot n^{O(1)}$-time algorithms) assuming the exponential time hypothesis (ETH) [11]. Further, as observed by Flum and Grohe [9], the sunflower lemma of Erdős and Rado [8] can be used to produce polynomial kernels for H-free vertex deletion problems.

Even restricted to this very simple case, edge modification problems remain elusive. Significant efforts have been devoted to an ongoing program that tries to answer these questions in a systematic way, and promising progress has been reported in literature. Recently, Aravind et al. [1] gave a complete answer to the first question: The H-free editing problem is NP-complete if and only if H contains at least three vertices. They also excluded subexponential-time algorithms for the NP-complete H-free edge modification problems, assuming ETH. Noting that H-free edge modification problems can always be solved in $2^{O(k)} \cdot n^{O(1)}$ time [2], we are left with the third problem, the existence of polynomial kernels.

Some of the H-free graph classes are important for their own structural reasons, e.g., most notably, cluster graphs and cographs, which are P_3-free graphs and P_4-free graphs respectively; hence the edge modification problems toward them have been well-studied [5, 10]. (Note that edge modification problems to P_2-free graphs, i.e., independent sets, are trivial.) Given the simplicity of H-free edge modification problems, and the naive FPT algorithms for them, it may sound shocking that many of them do *not* admit polynomial kernels. Indeed, the earliest incompressibility results of graph modification problems, by Kratsch and Wahlström [12], are on H-free edge modification problems. Guillemot et al. [10] excluded polynomial kernels for H-free edge deletion problems when H is a path of length at least seven or a cycle of length at least four. An influential result of Cai and Cai [3] furnishes a dichotomy on the compressibility of H-free edge modification problems when H is a path, a cycle, or a 3-connected graph.

We tend to believe that H-free edge modification problems admitting polynomial kernels are the exceptions. Our exploration suggests that graphs on four vertices play the pivotal roles if we want to fully map the territory. Let $\overline{H}$ be the complement graph of H. Then the H-free edge deletion problem is equivalent to the $\overline{H}$-free edge completion problem, while the edge editing problems are the same for H-free and $\overline{H}$-free graphs. We are thus focused on the four-vertex graphs (Figure 1); see Table 1 for a summary of compressibility results of H-free edge modification problems when H is one of them. We conjecture that H-free edge modification problems, when H being claw or paw, admit polynomial kernels.



(a) P_4 **(b)** C_4 **(c)** K_4 **(d)** claw **(e)** paw **(f)** diamond

Figure 1 Graphs on four vertices (their complements are omitted).

Table 1 The compressibility results of H-free edge modification problems for H being four-vertex graphs. Every result holds for the complement $\overline{H}$; e.g., the answers are also no when H is $2K_2$.

H	deletion	completion	editing
K_4	$O(k^4)$ [4]	trivial	$O(k^4)$ [4]
P_4	$O(k^3)$ [10]	$O(k^3)$ [10]	$O(k^3)$ [10]
diamond	$O(k^3)$ [14]	trivial	$O(k^8)$ [this paper]
claw	unkown	unkown	unkown
paw	unkown	unkown	unkown
C_4	no [10]	no [10]	no [10]

We show a polynomial kernel for the diamond-free editing problem. Our observations also lead to a cubic-vertex kernel for the diamond-free edge deletion problem, which is far simpler than the previous kernel of the same size [14].

Our key observations are on maximal cliques. A graph G is diamond-free if and only if every two maximal cliques of G share at most one vertex. We say that a maximal clique is of *type* I if it shares an edge with another maximal clique, or *type* II otherwise. It is not hard to see that to make a graph diamond-free, we should never delete edges from a sufficiently large clique. We thus put the maximal cliques of G into three categories, small type I, big type I, and type II. It turns out that a vertex participates in a diamond if and only if it is in a maximal clique of type I, and the small type-I maximal cliques are crucial for the problem.

The first phase of our algorithm comprises two routine reduction rules. If a (non-)edge participates in $k + 1$ or more diamonds that pairwise share only this (non-)edge, then it has to be in a solution. (This is exactly the reason why no edge is deleted from a "large" clique.) If there exists such an edge/non-edge, we delete/add it. We may henceforth assume that these two simple rules have been exhaustively applied. We are able to show that the ends of an edge added by a minimum solution must be from some small type-I maximal cliques. The situation for deleted edges is slightly more complex. The two ends of a deleted edge are either in a small type-I maximal clique, or in a type-II maximal clique. In the second case, the maximal clique has to intersect some small type-I maximal clique.

The second phase of our algorithm uses three nontrivial reduction rules to delete irrelevant vertices. To analyze the size of the kernel, we bound the number of vertices that are (a) in small type-I maximal cliques only, (b) in big type-I maximal cliques but not in any small type-I maximal clique, and (c) only in type-II maximal cliques. First, we show an upper bound on the number of type-I maximal cliques. This immediately bounds the number of vertices in part (a), because each small type-I maximal clique has a bounded size. For part (b), the focus now is to bound the sizes of big maximal cliques of type I. We introduce another reduction rule to delete certain "private vertices" from them. On the other hand, the pattern of vertices shared by big maximal cliques is very limited. We are thus able to bound the number of vertices in part (b), and we are left with part (c). We correlate a maximal clique K of type II with small maximal cliques of type I: we would touch K only because it had become type I after some operation, and this operation has to be an edge addition. Recall that an edge can only be added between two vertices in part (a). For each pair of them, we can build a blocker of $O(k^2)$ vertices from part (c). One more reduction rule is introduced to remove all vertices behind the blockers. Together with the bound of vertices in part (a), this bounds the number of vertices in part (c). They together give our main result.

▶ **Theorem 1.** *The diamond-free editing problem has a kernel of $O(k^8)$ vertices.*

2 Maximal cliques

All graphs discussed in this paper are undirected and simple. A graph G is given by its vertex set $V(G)$ and edge set $E(G)$. The *neighborhood* of a vertex v in a graph G, denoted by $N_G(v)$, consists of all the vertices adjacent to v in G. We extend this to a set $S \subseteq V(G)$ of vertices by defining the neighborhood $N_G(S)$ of S as $(\bigcup_{v\in S} N_G(v)) \setminus S$. For a set $U \subseteq V(G)$ of vertices, we denote by $G[U]$ the subgraph induced by U, whose vertex set is U and whose edge set comprises all edges of G with both ends in U. We use $G - v$, where v is a vertex of G, as a shorthand for $G[V(G) \setminus \{v\}]$. In a diamond, we refer to the edge between the two degree-three vertices as the *cross edge*, and the only non-edge the *missing edge*.

For a set E_+ of edges, we denoted by $G + E_+$ the graph obtained by adding edges in E_+ to G,– its vertex set is still $V(G)$ and its edge set becomes $E(G) \cup E_+$. The graph $G - E_-$ is defined analogously. Throughout the paper we always tacitly assume $E_+ \cap E(G) = \emptyset$ and $E_- \subseteq E(G)$; hence E_+ and E_- are disjoint. A *solution* of an instance (G, k) consists of a set E_+ of added edges and a set E_- of deleted edges such that $G + E_+ - E_-$ is diamond-free and $|E_+ \cup E_-| \leq k$. We use $E_\pm$ as a shorthand for $E_+ \cup E_-$, and there should be no ambiguities: $E_+ = E_\pm \setminus E(G)$ and $E_- = E_\pm \cap E(G)$. We also use $G \triangle E_\pm$ as a shorthand for $G + E_+ - E_-$.

We start from two routine reduction rules for edge editing problems. The correctness of them is straightforward: If we do not add/delete uv, then we have to delete/add at least $k + 1$ edges.

▶ **Rule 1.** *If there exist a non-edge uv and $2k + 2$ distinct vertices $x_1, y_1, \ldots, x_{k+1}, y_{k+1}$ in $N(u) \cap N(v)$ such that $x_i y_i \in E(G)$ for $1 \leq i \leq k + 1$, then add uv and decrease k by one.*

▶ **Rule 2.** *If there exist an edge uv and $2k + 2$ distinct vertices $x_1, y_1, \ldots, x_{k+1}, y_{k+1}$ in $N(u) \cap N(v)$ such that $x_i y_i \notin E(G)$ for $1 \leq i \leq k + 1$, then delete uv and decrease k by one.*

Whether Rule 1 (resp., Rule 2) is applicable to uv can be decided by finding a maximum matching in $G[N(u) \cap N(v)]$ (resp., the complement graph of $G[N(u) \cap N(v)]$). Therefore, Rules 1 and 2 can be applied in polynomial time. We call an instance (G, k) *reduced* if neither Rule 1 nor 2 is applicable to it. In the rest, we will focus on reduced instances. A similar idea as the two rules enables us to exclude some (non-)edges from consideration.

▶ **Proposition 2.** *A (non-)edge uv cannot be in a solution $E_\pm$ of a yes-instance (G, k), if*
(i) *$uv \in E(G)$ and there are $k + 1$ pairwise adjacent vertices in $N(u) \cap N(v)$; or*
(ii) *$uv \notin E(G)$ and there are $k + 1$ pairwise nonadjacent vertices in $N(u) \cap N(v)$.*

▶ **Proposition 3.** *Let (G, k) be a reduced yes-instance. For any (non-)edge uv in a solution of (G, k), the cardinality of $N(u) \cap N(v)$ is at most $3k$.*

Proof. We consider only $uv \in E_-$, and the argument for $uv \in E_+$ is similar and omitted. Let $W = N(u) \cap N(v)$; we find a maximum matching in the complement graph of $G[W]$, and let W' be the ends of the edges in the matching. Since Rule 2 is not applicable (to uv), $|W'| \leq 2k$. There cannot be non-edges between vertices in $W \setminus W'$; then by Proposition 2(i), the size of $W \setminus W'$ is at most k. Therefore, $|W| \leq 3k$. ◀

Our algorithm will be mostly concerned with maximal cliques. According to Proposition 2(i), a maximal clique on $k+3$ or more vertices cannot be touched by a minimum solution "from inside," but it may be touched "from outside". We call a maximal clique *big* if it contains at least $3k+2$ vertices, and *small* otherwise.

▶ **Lemma 4.** *Let (G, k) be a reduced instance.*

(i) *Two big maximal cliques of G share at most one vertex.*

(ii) *If (G, k) is a yes-instance, then a big maximal clique of G remains a maximal clique after applying a solution to (G, k).*

Proof. Let K_1 and K_2 be two big maximal cliques of G. Suppose first that some vertex $u \in K_1 \setminus K_2$ is adjacent to more than $2k+1$ vertices in K_2. Since K_2 is a maximal clique, we can find $v \in K_2 \setminus K_1$ nonadjacent to u, but then Rule 1 would be applicable (to uv). Hence, every vertex in $K_1 \setminus K_2$ has at most $2k+1$ neighbors in K_2, which implies $|K_1 \cap K_2| \leq 2k+1$. By assumption, $|K_1| \geq 3k+2$ and $|K_2| \geq 3k+2$. For each vertex in $K_1 \setminus K_2$, we can find $k+1$ non-neighbors in $K_2 \setminus K_1$. Therefore, we can greedily find $k+1$ pairs of distinct vertices $\{x_1, y_1\}$, $\ldots$, $\{x_{k+1}, y_{k+1}\}$ such that for all $1 \leq i \leq k+1$, (a) $x_i \in K_1 \setminus K_2$ and $y_i \in K_2 \setminus K_1$; and (b) $x_i y_i \notin E(G)$. Rule 2 would be applicable (to any edge in $G[K_1 \cap K_2]$) if $|K_1 \cap K_2| \geq 2$. Therefore, $|K_1 \cap K_2| \leq 1$, and this concludes the proof for assertion (i).

Let $E_\pm$ be a solution to (G, k) and $G^* = G \triangle E_\pm$. By Proposition 2(i), a big maximal clique K in G remains a clique in G^*. Let $v \in V(G) \setminus K$ and let $u \in K \setminus N_G(v)$. Since Rule 1 is not applicable to uv, there are at most $2k+1$ neighbors of v in K. Since $|K| \geq 3k+2$, at least one vertex in K remains nonadjacent to v in G^* because $|E_+| \leq k$. Therefore, K is a maximal clique in G^* as well. ◀

It is well known that a graph is diamond-free if and only if every pair of adjacent vertices is contained in exactly one maximal clique. (Proposition 5 implies this fact.) We say that a maximal clique is of *type* I if it shares two or more vertices with some other maximal clique, and *type* II otherwise. We can then rephrase the first sentence of this paragraph as: A graph is diamond-free if and only if it has no type-I maximal clique.

We use $\mathcal{K}_b(G)$, $\mathcal{K}_s(G)$, and $\mathcal{K}_2(G)$ to denote, respectively, the set of big maximal cliques of type I, the set of small maximal cliques of type I, and the set of maximal cliques of type II, of G. A maximal clique in G is in precisely one of them.

▶ **Proposition 5.** *(i) A maximal clique is of type* I *if and only if it contains both ends of the cross edge of a diamond. (ii) A vertex is in a maximal clique of type* I *if and only if it is contained in an induced diamond.*

Proof. The following argument proves assert (i), and it also works for assert (ii).

Let u, v, x, y be four vertices inducing a diamond in G with cross edge uv. We can find two maximal cliques K_1, K_2 containing u, v, x and u, v, y respectively. For any maximal clique K containing u, v, at least one of K_1, K_2 is different from K, hence K is of type I.

We now consider the "only if" direction. Let K_1 be a maximal clique of type I; by definition, there is another maximal clique K_2 such that $|K_1 \cap K_2| \geq 2$. For any vertex $x \in K_1 \setminus K_2$ and any vertex $u \in K_1 \cap K_2$, we can find another vertex $v \in K_1 \cap K_2$ different from u and a vertex $y \in K_2 \setminus K_1$ not adjacent to x (because K_2 is maximal). Clearly, these four vertices induce a diamond with cross edge uv. ◀

The following two statements help us understand edges added by a minimum solution.

▶ **Proposition 6.** *Let G be a diamond-free graph, and let $U \subseteq V(G)$ such that every vertex in $V(G) \setminus U$ is adjacent to at most one vertex of U. If $G[U] \triangle E_{\pm}$ is diamond-free for a set E_+ of non-edges in $G[U]$ and a set E_- of edges in $G[U]$, then so is $G \triangle E_{\pm}$.*

Proof. Suppose for contradiction that $G^* = G \triangle E_{\pm}$ contains a diamond; let D be a set of vertices inducing a diamond in G^*. Since $G[D]$ is not a diamond, at least one (non-)edge of this diamond belongs to $E_{\pm}$, and is between vertices of U. On the other hand, $G[U] \triangle E_{\pm}$ remains diamond-free, hence $D \not\subseteq U$. Therefore, $|D \cap U|$ is either two or three, but then a vertex in $D \setminus U$ is adjacent to at least two vertices of $D \cap U$ in G, a contradiction. ◀

▶ **Lemma 7.** *Let $E_{\pm}$ be a minimum solution to a reduced yes-instance (G, k). Every vertex incident to some edge in E_+ is contained in some small maximal clique of type* I *in G.*

Proof. Let $G^* = G \triangle E_{\pm}$, where uv is an edge in E_+, and let U be a maximal clique of G^* containing u, v. We argue first that v is in some induced diamond in $G[U]$.

Suppose for contradiction that v participates in no diamond in $G[U]$. Let $X = N_G(v) \cap U$. The subgraph $G[X]$ is a disjoint union of cliques: An induced path of length two would make a diamond with v. Let $\{A_1, \ldots, A_p\}$ be those nontrivial cliques (containing more than one vertex) in $G[X]$; let B be the other vertices of X; and let $C = U \setminus N_G[v]$. Then $\{A_1, \ldots, A_p, B, C\}$ is a partition of the set $U \setminus \{v\}$. Note that p or $|B|$ may be 0, but $|C| > 0$ because $u \in C$. To arrive at a contradiction, we will construct a solution $E'_{\pm}$ for $G[U]$ whose size is smaller than the number of non-edges in $G[U]$. Assume such an $E'_{\pm}$ exists and let G' be the graph obtained from G^* by replacing $G^*[U]$ with $G[U] \triangle E'_{\pm}$. Since U is a type-II maximal clique of G^*, for each $x \in V(G) \setminus U$ we have $|N_{G^*}(x) \cap U| \leq 1$. By Proposition 6, G' is diamond-free. This would however imply a strictly smaller solution than $E_{\pm}$, contradicting that $E_{\pm}$ is a minimum solution of (G, k). Now we show how to construct $E'_{\pm}$.

Case 1, $|B| \geq |C|$. We set $E'_+ = \emptyset$ and E'_- the set of edges in $G[C]$. No edge in E'_- is incident to v or $N(v)$, and hence $N(v) \cap U$ is still a disjoint union of cliques in G'. On the other hand, no vertex $x \in C$ is in any diamond in $G'[U]$ because $N_{G'}(x) \cap U$ is an independent set. Thus, $G'[U]$ is diamond-free. Since B is an independent set of G, and v is nonadjacent to C, we have $|E_+ \cap U^2| \geq \binom{|B|}{2} + |C| \geq \binom{|C|}{2} + |C| > |E'_-| = |E'_+ \cup E'_-|$.

Case 2, $|B| < |C|$. We set E'_+ to be the set of non-edges in $G[B \cup C]$, and E'_- the set of edges between $B \cup C$ and $U \setminus (B \cup C)$. To see that $G'[U]$ is diamond-free, note that its maximal cliques are $B \cup C$ and $\{v\} \cup A_i$ for $1 \leq i \leq p$, whose intersection is either $\{v\}$ or empty. We then calculate the cardinality of $E_+ \cap U^2$, which comprises three parts, those among $B \cup C$, which is exactly E'_+, those between C and v, and those between C and A_i's. Since v does not belong to any diamond in $G[U]$, each vertex in C is adjacent to at most one vertex of $A_i, 1 \leq i \leq p$. In other words, for each $x \in C$ and each $1 \leq i \leq p$, the number of non-edges between x and A_i is at least one. Therefore $|E_+ \cap U^2| \geq |E'_+| + |C| + |C| \times p > |E'_+| + |B| + |C| \times p \geq |E'_+| + |E'_-|$.

Now that v is in some induced diamond in $G[U]$, we can find a maximal clique K of G containing v and another two vertices of this diamond. Since U is not a clique in G, we have $K \neq U$. And $|K \cap U| \geq 3$ implies that K cannot induce a maximal clique of G^*. Hence by Lemma 4(ii), it is small. This concludes the proof of the lemma. ◀

After delimiting the ends of the edges added by a minimum solution, we now turn to the ends of those edges deleted by a minimum solution. The next lemma states that some maximal cliques in G remain maximal cliques after applying the solution.

▶ **Lemma 8.** *Let $E_{\pm}$ be a minimum solution to an instance (G, k), and let K be a maximal clique of type* II *in G. If E_+ contains neither (i) an edge between $u \in K$ and $v \in N(K)$, nor (ii) two edges between vertices of K and the same vertex in $V(G) \setminus K$, then K remains a maximal clique (of type* II*) in $G \triangle E_{\pm}$.*

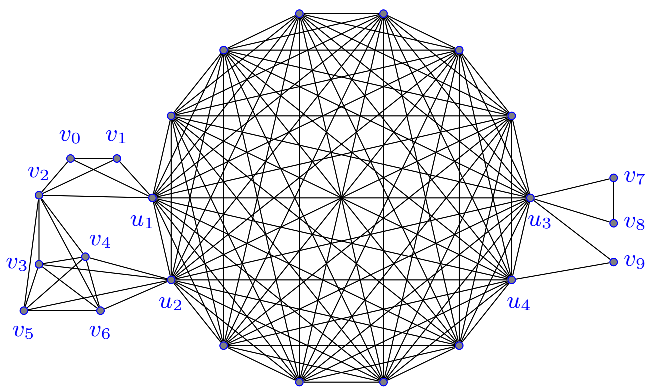


Figure 2 An example with $k = 4$, of which a minimum solution is $\{+u_2v_2, -u_1v_2, -v_0v_1, -u_3v_9\}$. (Note that u_1v_2 and v_0v_1 are not in any diamond of G.) It has six maximal cliques, $K_1 = \{v_0, v_1, v_2, u_1\}$, $K_2 = \{v_2, v_3, v_4, v_5, v_6\}$, $K_3 = \{u_2, v_3, v_4, v_5, v_6\}$, $K_4 = \{u_3, v_7, v_8\}$, $K_5 = \{u_3, u_4, v_9\}$, while K_6 comprises of u_1, u_2, u_3, u_4 and other ten unlabeled vertices. Four of these maximal cliques, K_2, K_3, K_5, and K_6, are of type I, of which only K_6 is big, the other two of type II. All 14 labeled vertices are vulnerable, and the other 8 unlabeled vertices are guarded.

Proof. Let $G^* = G \triangle E_\pm$. Since K is a type-II maximal clique of G, each vertex $v \in V(G) \setminus K$ has at most one neighbor in K. By the assumption that E_+ contains neither (i) nor (ii), this remains true in $G + E_+$ and G^*. On the other hand, E_- cannot contain edges of $G[K]$; otherwise, by Proposition 6, G^* remains diamond-free after replacing $G^*[K]$ by $G[K]$, which implies a strictly smaller solution than $E_\pm$. Therefore, K is a maximal clique in G^*. ◀

The next corollary follows from Lemma 7 and Lemma 8.

▶ **Corollary 9.** *Let $E_\pm$ be a minimum solution to a reduced yes-instance (G, k), and let K be a maximal clique of G containing both ends of an edge in E_-. Then either $K \in \mathcal{K}_s(G)$, or $K \in \mathcal{K}_2(G)$ and K intersects one clique in $\mathcal{K}_s(G)$.*

Lemma 7 and Corollary 9 motivate the following definitions. A vertex v is *vulnerable* in graph G if (1) there exists some $K \in \mathcal{K}_s(G)$ containing v; or (2) there are intersecting maximal cliques $K_1 \in \mathcal{K}_s(G)$ and $K_2 \in \mathcal{K}_2(G)$ such that $v \in K_2$. A vertex is *guarded* if it is not vulnerable. Lemma 7 and Corollary 9 can be summarized as: No (non-)edge in a minimum solution can be incident to a guarded vertex. See Figure 2 for an illustration.

3 The kernel

We partition the vertex set of a reduced graph into five parts, and deal with them separately.

(i) vertices in small maximal cliques of type I (all of them are vulnerable);
(ii) vulnerable vertices in big maximal cliques of type I but not in the previous part;
(iii) other vulnerable vertices (not in any maximal cliques of type I);
(iv) guarded vertices in (big) maximal cliques of type I; and
(v) other guarded vertices (not in any maximal cliques of type I).

Note that for this purpose we do *not* need to enumerate the maximal cliques. The key observation is that we can easily find the cross edges of all diamonds by enumeration, from which we can identify all vertices and edges in maximal cliques of type I. We use the procedure `partition` presented in Algorithm 1, which computes this partition in three steps: It first finds all vertices in a maximal clique of type I, from which it identifies those in a small maximal clique of type I, and finally it uses them to get all vulnerable vertices.

Algorithm 1 The procedure `partition`.

```
INPUT: a reduced instance $(G, k)$.
OUTPUT: vertices in the five parts have (i) mark “small,” (ii) marks “vulnerable” and
“type I,” (iii) mark “vulnerable,” (iv) mark “type I,” and (v) no mark, respectively.

1. for each edge $uv \in E(G)$ where $N(u) \cap N(v)$ does not induce a clique do
1.1.     mark $uv$ “cross edge”;
1.2.     mark $u, v$ and all vertices in $N(u) \cap N(v)$ as “type I”;
1.3.     mark all edges between these vertices as “type I”;
    \\a vertex is in a maximal clique of type I if and only if it is marked “type I.”
2. for each marked vertex $v$ do
2.1.     if $G[N(v)]$ is not a cluster (a disjoint union of cliques) do mark $v$ “small”;
2.2.     else if a clique in $N(v)$ of size $\leq 3k$ contains a cross edge do mark $v$ “small”;
3. for each unmarked edge $uv \in E(G)$ do
3.1.     find the maximal clique $K$ containing $u$ and $v$;
3.2.     if $K$ contains any vertex marked “small” then mark vertices in $K$ “vulnerable”;
3.3.     mark every edge in $K$ “checked.”
```

It is easy to check that procedure `partition` runs in polynomial time. We now show its correctness.

▶ **Lemma 10.** *Procedure partition is correct.*

Proof. An edge $uv \in E(G)$ is a cross edge if and only if $N(u) \cap N(v)$ does not induce a clique; this justifies step 1.1. Steps 1.2 and 1.3 follow from Proposition 5(i).

Step 2 considers all vertices in maximal cliques of type I. If some component of $G[N(v)]$ is not a clique, we can find a path xyz of length two. There are two different maximal cliques containing v, x, y and v, y, z respectively. Both are of type I, and hence by Lemma 4(i), at least one of them is small. Step 2.2 also follows from Proposition 5(i). If a vertex v is not marked in step 2, then every maximal clique containing v is either big or of type II. Therefore, all vertices in small maximal cliques of type I have been correctly identified in step 2.

Step 3 finds other vulnerable vertices. By definition, such a vertex is in some maximal clique of type II. If a type-II maximal clique consists of an isolated vertex, it is guarded and not marked in step 3. We may hence consider only nontrivial maximal cliques. All edges in a type-II maximal clique remain unmarked. Note that any two vertices of a type-II maximal clique determines this clique: It is the only maximal clique that contains these two vertices. Vertices in the clique are vulnerable if and only if it contains a vertex marked “small.” We only need to check the clique K once, so we mark them to avoid unnecessary repetition in step 3.3. After step 3, all type-II maximal cliques have been checked. ◀

3.1 Maximal cliques of type i

We start from the vertices in some small type-I maximal cliques, and denote them by $S(G)$, i.e., $S(G) = \bigcup_{K \in \mathcal{K}_s(G)} K$. Noting that the final graph has no small type-I maximal cliques, we can bound the size of $S(G)$ by relating vertices in it to edges in a minimum solution.

▶ **Lemma 11.** *If (G, k) is a reduced yes-instance, then $|S(G)| \leq 18k^3 + 2k$.*

Proof. Let $E_\pm$ be a minimum solution of (G, k). Let $X = \bigcup_{xy \in E_\pm} \{x, y\}$ and $Y = \bigcup_{xy \in E_\pm} N_G(x) \cap N_G(y)$, i.e., all vertices incident to a (non-)edge in the solution and respectively, all vertices that is a common neighbor of the two ends of a (non-)edge in the solution. Note that $|X| \leq 2k$, and by Proposition 3, $|Y| \leq 3k \cdot |E_\pm| \leq 3k^2$. Since $|S(G) \cap (X \cup Y)| \leq |X \cup Y| \leq 3k^2 + 2k$, it suffices to bound $S(G) \setminus (X \cup Y)$. A vertex $v \in S(G) \setminus (X \cup Y)$ cannot be contained in two type-I maximal cliques of G if they share more than one vertex: Otherwise, there is a diamond (as in Proposition 5(ii)) in $N_G[v]$, but then v has to be in $X \cup Y$, a contradiction.

Let us now consider the set of small type-I maximal cliques of G that contain vertices from $S(G) \setminus (X \cup Y)$, which we denote by $\mathcal{K}'$. We argue by contradiction that any pair of cliques in $\mathcal{K}'$ shares at most one vertex. Suppose otherwise, there are two maximal cliques $K_1, K_2 \in \mathcal{K}'$ with $|K_1 \cap K_2| \geq 2$. We have seen that $K_1 \cap K_2$ is disjoint from $S(G) \setminus (X \cup Y)$. Now let $u \in K_1 \setminus K_2$ and $v \in K_2 \setminus K_1$ be two vertices in $S(G) \setminus (X \cup Y)$. Then there is a diamond with u and two vertices in $K_1 \cap K_2$ and one vertex in $K_2 \setminus K_1$. But by the assumption $u \notin X \cup Y$, we cannot add or delete any edge incident to u; on the other hand, $v \notin X \cup Y$ forbids the deletion of other three edges, a contradiction.

Let $v \in S(G) \setminus (X \cup Y)$, and let K be a clique in $\mathcal{K}'$ containing v. By definition, there exists a diamond in which (1) v is a degree-two vertex; (2) the two degree-three vertices are in K; and (3) the other degree-two vertex is not in K. Since v is not in $X \cap Y$, one of the two edges of this diamond that are incident to the other degree-two vertex has to be in E_-. In other words, K contains for some edge $xy \in E_-$, one in $\{x, y\}$ and a common neighbor of x, y. By Proposition 3, for each edge $xy \in E_-$, there are at most $3k$ vertices in $N_G(x) \cap N_G(y)$; for each $z \in N_G(x) \cap N_G(y)$, there can be at most one clique in $\mathcal{K}'$ containing x, z and at most one clique in $\mathcal{K}'$ containing y, z. Therefore, there can be at most $3k \cdot 2 \cdot |E_-| \leq 6k^2$ cliques in $\mathcal{K}'$. By definition, each clique in it is small and has at most $3k + 1$ vertices, of which at least two are not in $S(G) \setminus (X \cup Y)$. Hence $|S(G) \setminus (X \cup Y)| \leq (3k - 1) \cdot 6k^2 = 18k^3 - 6k^2$. Putting the two parts together, we have $|S(G)| \leq 18k^3 + 2k$. ◀

Next, we consider the big type-I maximal cliques, and bound first the number of them.

▶ **Lemma 12.** *If (G, k) is a reduced yes-instance, then $|\mathcal{K}_b(G)| \leq 6k^2$.*

Proof. By Lemma 4, the only way to transform a big maximal clique of type I into one of type II is deleting edges incident to it. For an edge $e = uv \in E_-$, denote by $\mathcal{K}_e$ the set of big type-I maximal cliques containing one in $\{u, v\}$, and one vertex in $N(u) \cap N(v)$. Note that $\mathcal{K}_b(G) = \bigcup_{e \in E_-} \mathcal{K}_e$. By Proposition 3, $\mathcal{K}_e$ has at most $6k$ maximal cliques. Then $|\mathcal{K}_b(G)| \leq 6k \cdot |E_-| = 6k^2$. ◀

To bound the size of big type-I maximal clique, we introduce another reduction rule.

▶ **Rule 3.** *Let $K \in \mathcal{K}_b(G)$ with $|K| \geq 3k + 3$. If K contains a guarded vertex x that does not occur in any other type-I maximal clique of G, delete it.*

▶ **Lemma 13.** *Rule 3 is safe: A reduced instance (G, k) is a yes-instance if and only if $(G - x, k)$ is a yes-instance.*

Proof. It is easy to see that $(G - x, k)$ is a reduced instance, and every solution of (G, k) confined to $G - x$ is a solution of $(G - x, k)$. For the "if" direction, let $E_\pm$ be a minimum solution of $(G - x, k)$, and let $G^* = G \triangle E_\pm$. Note that $(G - x) \triangle E_\pm = G^* - x$, and it is diamond-free. No edge in $E_\pm$ is incident to x, and hence $N_G(x) = N_{G^*}(x)$, which we simply denote by $N(x)$. By Proposition 5(ii), it suffices to prove that each maximal clique of G^* containing x is of type II. For this purpose, we show that each component of $G[N(x)]$ is either a single vertex or a type-II maximal clique in $G^* - x$.

Note that $K \setminus \{x\}$ is a big maximal clique in $G - x$: It is a clique of size at least $3k + 2$, and its maximality follows from Lemma 4(i). Hence, by Lemma 4(ii), $K \setminus \{x\}$ is a maximal clique (of type II) in $G^* - x$. Since x is a guarded vertex that does not occur in any other type-I maximal clique, every other maximal clique K' containing x in G is of type II, and it cannot intersect any small type-I maximal clique. Therefore, by Lemma 7, no edge added by E_+ can be incident to any vertex in $N(x)$. From Lemma 8 we can conclude that $K' \setminus \{x\}$ either contains only a vertex or is a maximal clique (of type II) in $G^* - x$.

Since no edge added by E_+ is between two vertices in $N(x)$ and since x is a guarded vertex, each component of $G[N(x)]$ is either $K \setminus \{x\}$ or $K' \setminus \{x\}$, hence is either a single vertex or a type-II maximal clique in $G^* - x$. This concludes the proof. ◀

▶ **Lemma 14.** *Let (G, k) be a reduced yes-instance. If Rule 3 is not applicable, then for each $K \in \mathcal{K}_b(G)$, we have that $|K| = O(k^3)$.*

Proof. Without loss of generality, assume that $|K| \geq 3k + 3$. Since Rule 3 is not applicable, every vertex in K is either a vulnerable vertex, or a guarded vertex in more than one big type-I maximal clique. Let U_1 and U_2 be the set of vulnerable vertices in $K \cap S(G)$ and $K \setminus S(G)$ respectively. By the definition, each vertex in U_2 is adjacent to some vertex in $S(G) \setminus U_1$ by an edge of type-II maximal clique. For each vertex $v \in S(G) \setminus U_1$, the cardinality of $U_2 \cap N(v)$ is at most one; otherwise, there is a type-I maximal clique containing $U_2 \cap N(v)$ and v which by Lemma 4(i) is small, contradicting to $U_2 \subseteq K \setminus S(G)$. Therefore, $|U_2| \leq |S(G) \setminus U_1|$, and by Lemma 11, K contains at most $18k^3 + 2k$ vulnerable vertices. By Lemma 4(i), every pair of big type-I maximal cliques shares at most one vertex. Hence, by Lemma 12, K contains at most $6k^2$ guarded vertices that appear in some other big maximal cliques of type I. Putting them together we get $|K| \leq 18k^3 + 2k + 6k^2$. ◀

The next corollary follows immediately from Lemmas 12 and 14.

▶ **Corollary 15.** *Let (G, k) be a reduced yes-instance. If Rule 3 is not applicable, then the number of vertices that are contained in some cliques in $\mathcal{K}_b(G)$ is $O(k^5)$.*

3.2 Maximal cliques of type ii

We have bounded the number of vertices in all maximal cliques of type I, and it remains to bound the number of vertices that occur *only* in maximal cliques of type II. Let $T(G)$ denote these vertices, i.e., $T(G) = V(G) \setminus \bigcup_{K \in \mathcal{K}_s(G) \cup \mathcal{K}_b(G)} K$. It may not be surprising that we can delete all the guarded vertices in them.

▶ **Rule 4.** *If there is a guarded vertex x not in any type-I maximal clique of G, delete it.*

▶ **Lemma 16.** *Rule 4 is safe: A reduced instance (G, k) is a yes-instance if and only if $(G - x, k)$ is a yes-instance.*

Proof. It is easy to see that $(G - x, k)$ is a reduced instance, and every solution of (G, k) confined to $G - x$ is a solution of $(G - x, k)$. For the other direction, let $E_\pm$ be a minimum solution of $(G - x, k)$, and it is sufficient to show that x is not part of any diamond in $G^* = G \triangle E_\pm$. Note that x is a vertex which is part of only type-II maximal cliques in G and not adjacent to any vertex in small type-I maximal cliques in G. Therefore, by Lemma 7, none of the vertices in $N(x)$ is incident to any edges of E_+. If x is part of a diamond in G^*, then it is formed by a deletion of an edge in $G[N[x]]$ by E_-. But this is not possible by Corollary 9, as none of the edges in $G[N[x]]$ is part of any type-II maximal clique which intersects with a small type-I maximal clique in $G - x$. ◀

If Rule 4 is not applicable, then all vertices in $T(G)$ are vulnerable. As demonstrated in Figure 2, an edge may be deleted from a maximal clique of type II. In that example, neither end of the deleted edge v_0v_1 is in any maximal clique of type I. This can happen only *after* some modification happens in the neighborhood of this vertex – u_2v_2 added in the example. According to Proposition 2, however, this would not happen when $|K| \geq k+3$. In other words, to make sure a large clique in $\mathcal{K}_2(G)$ is immutable to future modifications, it suffices to keep $k+3$ of its vertices. This motivates the following reduction rule, whose statement is however more complex than previous ones. The main trouble here is that we are not allowed to delete all but $k+3$ guarded vertices from a clique in $\mathcal{K}_2(G)$, because it may be required for another clique in $\mathcal{K}_2(G)$. For a pair of vertices u, v, we denote by $N(u,v)$ the set of common neighbors of u and v not in $S(G)$, i.e., $N(u,v) = (N(u) \cap N(v)) \setminus S(G)$.

▶ **Proposition 17.** *Let u, v be two vertices in G. If $uv \notin E(G)$, then $N(u,v)$ form an independent set. Moreover, if $uv \in E_+$ for a solution $E_\pm$ of (G,k), then $|N(u,v)| \leq k$.*

Proof. If $G[N(u,v)]$ has an edge xy, then $\{u,v,x,y\}$ forms a diamond. There are two type-I maximal cliques containing $\{x,y,u\}$ and $\{x,y,v\}$ respectively. By Lemma 4(i), at least one of them is small, contradicting to $x,y \notin S(G)$. The second claim follows from Proposition 2. ◀

Our last rule would keep at most $k+1$ from such sets. To avoid unnecessary clutters, we simply say we mark $k+1$ vertices in $N(u,v)$, even if its size is smaller than $k+1$; in which case, we mark all of them.

▶ **Rule 5.** *For each pair of vertices $u, v \in S(G)$, arbitrarily mark $k+1$ vertices in $N(u,v)$. If $|N(u,v)| \leq k$, then for each vertex $w \in N(u,v)$, arbitrarily mark $k+1$ vertices in $N(u,w)$ and $k+1$ vertices in $N(v,w)$. If there is an unmarked vertex x in $T(G)$, delete it.*

▶ **Lemma 18.** *Rule 5 is safe: A reduced instance (G,k) is a yes-instance if and only if $(G-x,k)$ is a yes-instance.*

Proof. It is easy to see that $(G-x,k)$ is a reduced instance, and every solution of (G,k) confined to $G-x$ is a solution of $(G-x,k)$. For the "if" direction, let $E_\pm$ be a minimum solution of $(G-x,k)$, and let $G^* = G \triangle E_\pm$. We show that each maximal clique of G^* containing x is a maximal clique of G and is of type II in G^*. Since Proposition 5(ii) implies that deleting a vertex not in any type-I maximal clique does not alter type-I maximal cliques, we have $S(G') = S(G)$.

Let K be a maximal clique of G containing x; note that K is a maximal clique of type II in G, as $x \in T(G)$. We argue that $|N_{G^*}(y) \cap K| \leq 1$ for every $y \in V(G) \setminus K$. Since K is a maximal clique of type II in G, we have (1) $|N_G(y) \cap K|$ is either 0 or 1; and (2) for every pair of vertices $u, v \in K$, $N(u,v) \subseteq N_G(u) \cap N_G(v) = K$.

Suppose first that there are at least two edges between y and K in E_+. Let $u, v \in K$ be two vertices such that $yu, yv \in E_+$. Then by Lemma 7, $u, v \in S(G')$, and hence $u, v \in S(G)$. Clearly, $x \neq u$, $x \neq v$ and x is an unmarked vertex in $N(u,v)$. Further, there are $k+1$ marked vertices in $N(u,v)$. It follows that $|K \setminus \{x\}| \geq k+3$, and E_- does not have any edge in $G'[K \setminus \{x\}]$ by Proposition 2(i). Therefore, for each marked vertex $z \in N(u,v)$ that is not adjacent to y, the set $\{u,v,y,z\}$ induces a diamond in $G' + \{yu, yv\}$. The only edge we can edit is yz, but $|N_G(y) \cap K| \leq 1$, and there are at least $k+2$ edges between y and K, which is impossible.

Hence, at most one edge can be added between y and K by E_+. If $|N_G(y) \cap K| = 0$, or $|N_G(y) \cap K| = 1$ but the only edge between y and K is deleted, then it is trivial that y is adjacent to at most one vertex of K in G^*. Suppose that $N_{G^*}(y) \cap K = \{u,v\}$ while only u

is in $N_G(y)$; note that $yu \notin E_-$ and $yv \in E_+$. By Lemma 7, $y, v \in S(G')$, and hence in $S(G)$. According to Proposition 17, there are at most k vertices in $N(v, y)$ in G'. If $u \notin S(G)$, then it has been marked; hence $x \neq u$. Also, $x \neq v$ as $x \in T(G)$. By the rule, no matter whether u is in $S(G)$ or not, we should have marked vertices in $N(u, v)$. Since $x \in N(u, v)$ but is not marked, we have $|N(u, v)| > k+1$. Let z be any marked vertex in $N(u, v)$; it is not in $N_G(y)$ by assumption. But then $\{u, v, y, z\}$ induces a diamond in $G' + yv$, in which we have to add the missing edge yz, which requires $|E_+| > k$, a contradiction.

We have thus concluded $|N_{G^*}(y) \cap K| \leq 1$ for each vertex y in $V(G) \setminus K$. By Proposition 6, $K \setminus \{x\}$ remains a clique in $G^* - x$, otherwise we can find a strictly smaller solution. Then K is a maximal clique of type II in G^*. On the other hand, according to Proposition 17, no edge is added between two vertices of $N_G(x)$. Therefore, $N(x)$ induces exactly the same subgraph in G and G^*. Hence, any maximal clique of G^* containing x is a maximal clique of G as well, hence of type II in G^*. This concludes the proof of the lemma. ◀

Now Theorem 1 follows by counting numbers of different kinds of vertices.

Proof of Theorem 1. We show first that Rules 3–5 can be applied in polynomial time. For a guarded vertex x, $N(x)$ induces a cluster graph and each maximal clique in the cluster graph together with x forms the maximal cliques of G containing x. Recall that a maximal clique is of type I if and only if it contains both ends of a cross edge. Since the procedure `partition` finds all guarded vertices (no mark) and cross edges, we can find for each guarded vertex all type-I maximal cliques and type-II maximal cliques containing it in polynomial time. Therefore, both Rules 3 and Rule 4 can be applied in polynomial time. Moreover, the procedure `partition` finds all vertices in $S(G)$ (mark "small") and $T(G)$ (no mark "type I"), and hence Rule 5 can be applied in polynomial time.

We claim that if none of Rules 3–5 is applicable to a reduced yes-instance (G, k), then $|V(G)| = O(k^8)$. By Lemma 11, the number of vertices in small type-I maximal cliques is $|S(G)| = O(k^3)$. By Corollary 15, we have $O(k^5)$ vertices in big type-I maximal cliques. For each pair of vertices u, v in $S(G)$, we mark at most $k+1$ common neighbors of them. For each common neighbor w of u, v, we mark at most $2k+2$ vertices: $k+1$ vertices in $N(u, w)$ and $k+1$ vertices in $N(v, w)$. Hence $|T(G)| = O(k^8)$, and $|V(G)| = O(k^3)+O(k^5)+O(k^8) = O(k^8)$. ◀

4 A cubic kernel for diamond-free edge deletion

We use four simple reduction rules to get a cubic kernel for diamond-free edge deletion. The details of this section are omitted due to space limit.

1. If there exist an edge uv and $2k+2$ distinct vertices $x_1, y_1, \ldots, x_{k+1}, y_{k+1}$ in $N(u) \cap N(v)$ such that $x_i y_i \notin E(G)$ for $1 \leq i \leq k+1$, then delete uv and decrease k by one.
2. Mark an edge uv "permanent" if there are $2k+2$ distinct vertices $x_1, y_1, \ldots, x_{k+1}, y_{k+1}$ in $N(u) \cap N(v)$ such that $x_i y_i \in E(G)$ for all $1 \leq i \leq k+1$. If there exists a diamond consisting of only permanent edges, return a trivial no-instance.
3. If there is a vertex x not in any small maximal clique, delete it.
4. Delete all edges and vertices not in any maximal clique of type I.

▶ **Lemma 19.** *Let (G, k) be a yes-instance of the diamond-free edge deletion problem. If none of the reduction rules is applicable, then $|V(G)| = O(k^3)$.*

References

1 N. R. Aravind, R. B. Sandeep, and Naveen Sivadasan. Dichotomy results on the hardness of H-free edge modification problems. *SIAM Journal on Discrete Mathematics*, 31(1):542–561, 2017.

2 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letter*, 58(4):171–176, 1996.

3 Leizhen Cai and Yufei Cai. Incompressibility of H-free edge modification problems. *Algorithmica*, 71(3):731–757, 2015.

4 Yufei Cai. Polynomial kernelisation of H-free edge modification problems. Mphil thesis, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong SAR, China, 2012.

5 Yixin Cao and Jianer Chen. Cluster editing: Kernelization based on edge cuts. *Algorithmica*, 64(1):152–169, 2012.

6 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Undergraduate texts in computer science. Springer, 2013.

7 Ehab S. El-Mallah and Charles J. Colbourn. The complexity of some edge deletion problems. *IEEE Transactions on Circuits and Systems*, 35(3):354–362, 1988.

8 Paul Erdős and Richard Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 35(1):85–90, 1960.

9 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

10 Sylvain Guillemot, Frédéric Havet, Christophe Paul, and Anthony Perez. On the (non-) existence of polynomial kernels for P_l-free edge modification problems. *Algorithmica*, 65(4):900–926, 2013.

11 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

12 Stefan Kratsch and Magnus Wahlström. Two edge modification problems without polynomial kernels. *Discrete Optimization*, 10(3):193–199, 2013.

13 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

14 R. B. Sandeep and Naveen Sivadasan. Parameterized Lower Bound and Improved Kernel for Diamond-free Edge Deletion. In *10th International Symposium on Parameterized and Exact Computation*, volume 43 of *LIPIcs*, pages 365–376. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

15 Mihalis Yannakakis. Edge-deletion problems. *SIAM Journal on Computing*, 10(2):297–309, 1981.

Parallel and I/O-efficient Randomisation of Massive Networks using Global Curveball Trades

Corrie Jacobien Carstens
University of Amsterdam, Netherlands
c.j.carstens@uva.nl

Michael Hamann
Karlsruhe Institute of Technology, Germany
michael.hamann@kit.edu

Ulrich Meyer
Goethe University, Frankfurt, Germany
umeyer@ae.cs.uni-frankfurt.de

Manuel Penschuck
Goethe University, Frankfurt, Germany
mpenschuck@ae.cs.uni-frankfurt.de

Hung Tran
Goethe University, Frankfurt, Germany
htran@ae.cs.uni-frankfurt.de

Dorothea Wagner
Karlsruhe Institute of Technology, Germany
dorothea.wagner@kit.edu

Abstract

Graph randomisation is a crucial task in the analysis and synthesis of networks. It is typically implemented as an *edge switching* process (*ESMC*) repeatedly swapping the nodes of random edge pairs while maintaining the degrees involved [23]. Curveball is a novel approach that instead considers the whole neighbourhoods of randomly drawn node pairs. Its Markov chain converges to a uniform distribution, and experiments suggest that it requires less steps than the established *ESMC* [6]. Since trades however are more expensive, we study Curveball's practical runtime by introducing the first efficient Curveball algorithms: the I/O-efficient *EM-CB* for simple undirected graphs and its internal memory pendant *IM-CB*. Further, we investigate *global trades* [6] processing every node in a single super step, and show that undirected global trades converge to a uniform distribution and perform superior in practice. We then discuss *EM-GCB* and *EM-PGCB* for global trades and give experimental evidence that *EM-PGCB* achieves the quality of the state-of-the-art *ESMC* algorithm *EM-ES* [15] nearly one order of magnitude faster.

2012 ACM Subject Classification Mathematics of computing → Random graphs

Keywords and phrases Graph randomisation, Curveball, I/O-efficiency, Parallelism

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.11

Supplement Material Stable versions of *IM-CB* and *EM-GCB* are released as part of NetworKit (`http://network-analysis.info`).

Funding This work was partially supported by Deutsche Forschungsgemeinschaft (DFG) under grants ME 2088/3-2, ME 2088/4-2, and WA 654/22-2.

Acknowledgements We thank the anonymous reviewers for their many insightful comments and suggestions.

© Corrie Jacobien Carstens, Michael Hamann, Ulrich Meyer, Manuel Penschuck, Hung Tran, and Dorothea Wagner;
licensed under Creative Commons License CC-BY

1 Introduction

In the analysis of complex networks, such as social networks, the underlying graphs are commonly compared to random graph models to understand their structure [17, 27, 34]. While simple models like Erdős-Rényi graphs [11] are easy to generate and analyse, they are too different from commonly observed powerlaw degree sequences [27, 26, 34]. Thus, random graphs with the same degree sequence as the given graph are frequently used [8, 17, 32]. In practice, many of these graphs are simple graphs, i.e. graphs without self-loops and multiple edges. In order to obtain reliable results in these cases, the graphs sampled need to be simple since non-simple models can lead to significantly different results [31, 32]. The randomisation of a given graph is commonly implemented as an edge switching Markov chain *ESMC* [8, 24].

Nowadays, massive graphs that cannot be processed in the RAM of a single computer, require new analysis algorithms to handle these huge datasets. In turn, large benchmark graphs are required to evaluate the algorithms' scalability – in terms of speed and quality. LFR is a standard benchmark for evaluating clustering algorithms which repeatedly generates highly biased graphs that are then randomised [18, 19]. [15] presents the external memory LFR generator *EM-LFR* and its I/O-efficient edge switching *EM-ES*. Although *EM-ES* is faster than previous results even for graphs fitting into RAM, it dominates *EM-LFR*'s running time. Alternative sampling via the Configuration Model [25] was studied to reduce the initial bias and the number of *ESMC* steps necessary [14]. Still, graph randomisation remains a major bottleneck during the generation of these huge graphs.

The Curveball algorithm has been originally proposed for randomising binary matrices while preserving row and column sums [35, 36] and has been adopted for graphs [5, 6]: instead of switching a pair of edges as in *ESMC*, Curveball trades the neighbours of two nodes in each step. Carstens et al. further propose the concept of a *global trade*, a super step composed of single trades targetting every node[1] in a graph once [6]. The authors show that global trades in bipartite or directed graphs converge to a uniform distribution, and give experimental evidence that global trades require fewer Markov chain steps than single trades. However, while fewer steps are needed, the trades themselves are computationally more expensive. Since we are not aware of previous efficient Curveball algorithms and implementations, we investigate this trade-off here.

Our contributions. We present the first efficient algorithms for Curveball: the (sequential) internal memory and external memory algorithms *IM-CB*[2] and *EM-CB* for the Simple Undirected Curveball algorithm (see section 4). Experiments in section 5, indicate that they are faster than the established edge switching approaches in practice.

In section 3, we show that random global trades lead to uniform samples of simple, undirected graphs and demonstrate experimentally in section 5 that they converge even faster than the corresponding number of uniform single trades. Exploiting structural properties of global trades, we simplify *EM-CB* yielding *EM-GCB* and the parallel I/O-efficient *EM-PGCB* which achieves *EM-ES*'s quality nearly one order of magnitude faster in practice (see section 5).

[1] For an odd number n of nodes, either a single node is left out or equivalently an isolated node is added.
[2] We prefix internal memory algorithms with `IM` and I/O-efficient algorithms with `EM`. The suffices `CB`, `GCB`, and `PGCB` denote Curveball, CB. with global trades, and parallel CB. with global trades respectively.

2 Preliminaries and Notation

We define the short-hand $[k] := \{1, \dots, k\}$ for $k \in \mathbb{N}_{>0}$, and write $[\, x_i \,]_{i=a}^{b}$ for an ordered sequence $[x_a, x_{a+1}, \dots, x_b]$.

Graphs and degree sequences. A graph $G = (V, E)$ has $n = |V|$ sequentially numbered nodes $V = \{v_1, \dots, v_n\}$ and $m = |E|$ edges. Unless stated differently, graphs are assumed to be undirected and unweighted. To obtain a unique representation of an *undirected* edge $\{u, v\} \in E$, we use *ordered* edges $[u, v] \in E$ implying $u \le v$; in contrast to a directed edge, the ordering is used algorithmically but does not carry any meaning. A graph is called *simple* if it contains neither multi-edges nor self-loops, i.e. $E \subseteq \{\, \{u, v\} \mid u, v \in V \text{ with } u \neq v \,\}$. For node $u \in V$ define the *neighbourhood* $\mathcal{A}_u := \{v : \{u, v\} \in E\}$ and *degree* $\deg(u) := |\mathcal{A}_u|$. Let $d_{\max} := \max_v\{\deg(v)\}$ be the maximal degree of a graph. A vector $\mathcal{D} = [\, d_i \,]_{i=1}^{n}$ is a degree sequence of graph G iff $\forall v_i \in V\colon\ \deg(v_i) = d_i$.

Randomisation and Distributions. $\textsc{Pld}\,([a, b), \gamma)$ refers to an integer Powerlaw Distribution with exponent $-\gamma \in \mathbb{R}$ for $\gamma \ge 1$ and values from the interval $[a, b)$; let X be an integer random variable drawn from $\textsc{Pld}\,([a, b), \gamma)$ then $\mathbb{P}[X{=}k] \propto k^{-\gamma}$ (proportional to) if $a \le k < b$ and $\mathbb{P}[X{=}k] = 0$ otherwise. A statement depending on some number $x > 0$ is said to hold *with high probability* if it is satisfied with probability at least $1 - 1/x^c$ for some constant $c \ge 1$. Let S be a finite set, $x \in S$ and let σ be permutation on S, we define $\operatorname{rank}_\sigma(x)$ as the number of elements positioned in front of x by σ.

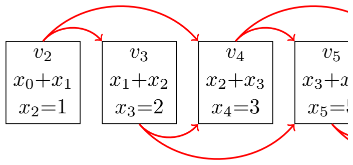
2.1 External-Memory Model

In contrast to classic models of computation, such as the unit-cost random-access machine, modern computers contain deep memory hierarchies ranging from fast registers, over caches and main memory to solid state drives (SSDs) and hard disks. Algorithms unaware of these properties may face performance penalties of several orders of magnitude.

We use the commonly accepted external memory (EM) model by Aggarwal and Vitter [1] to reason about the influence of data locality in memory hierarchies. It features two memory types, namely fast internal memory (IM or RAM) holding up to M data items, and a slow disk of unbounded size. The input and output of an algorithm are stored in EM while computation is only possible on values in IM. An algorithm's performance is measured in the number of I/Os required. Each I/O transfers a block of $B = \Omega(\sqrt{M})$ consecutive items between memory levels. Reading or writing n contiguous items is referred to as *scanning* and requires $\operatorname{scan}(n) := \Theta(n/B)$ I/Os. Sorting n consecutive items triggers $\operatorname{sort}(n) := \Theta((n/B) \cdot \log_{M/B}(n/B))$ I/Os. For all realistic values of n, B and M, $\operatorname{scan}(n) < \operatorname{sort}(n) \ll n$. Sorting complexity constitutes a lower bound for most intuitively non-trivial EM tasks [22]. EM queues use amortised $\mathcal{O}(1/B)$ I/Os per operation and require $\mathcal{O}(B)$ main memory [28]. An external priority queue (PQ) requires $\mathcal{O}(\operatorname{sort}(n))$ I/Os to push and pop n items [2].

2.2 TFP: Time Forward Processing

Time Forward Processing (*TFP*) is a generic technique to manage data dependencies of external memory algorithms [21]. Consider an algorithm computing values $x_1, \dots, x_n$ in which the calculation of x_i requires previously computed values. One typically models these dependencies using a directed acyclic graph $G{=}(V, E)$. Every node $v_i \in V$ corresponds to the computation of x_i and an edge $(v_i, v_j) \in E$ indicates that the value x_i is necessary to compute



```
PQ.push(<key=2, value=0>); PQ.push(<key=2, value=1>)
foreach i ← 2, ..., $n$ do
    sum ← 0
    while PQ.min.key == i do                    // Two iterations
        sum ← sum + PQ.remove-min().value
    print("$x_i$ =", sum)
    PQ.push(<key=i+1, sum>); PQ.push(<key=i+2, sum>)
```

Figure 1 Left: Dependency graph of the Fibonacci sequence (ignoring base case). **Right:** Time Forward Processing to compute sequence.

x_j. For instance consider the Fibonacci sequence $x_0 = 0,\ x_1 = 1,\ x_i = x_{i-1} + x_{i-2}\ \forall i \geq 2$ in which each node v_i with $i \geq 2$ depends on exactly its two predecessors (see Fig. 1). Here, a linear scan for increasing i suffices to solve the dependencies.

In general, an algorithm needs to traverse G according to some topological order $\prec_T$ of nodes V and also has to ensure that each v_j can access values from all v_i with $(v_i, v_j) \in E$. The *TFP* technique achieves this as follows: as soon as x_i has been calculated, messages of the form $\langle v_j, x_i \rangle$ are sent to all successors $(v_i, v_j) \in E$. These messages are kept in a minimum priority queue sorting the items by their recipients according to $\prec_T$. By construction, the algorithm only starts the computation v_i once all predecessors $v_j \prec_T v_i$ are completed. Since these predecessors already removed their messages from the PQ, items addressed to v_i (if any) are currently the smallest elements in the data structure and can be dequeued. Using a suited EM PQ [2], TFP incurs $\mathcal{O}(\text{sort}(k))$ I/Os, where k is the number of messages sent.

3 Randomisation schemes

Here, we summarise the randomisation schemes ESMC [24] and Curveball for simple undirected graphs [5], and then discuss the notion of global trades. Since these algorithms iteratively modify random parts of a graph, they can be analysed as finite Markov chains. It is well known that any finite, irreducible, aperiodic, and symmetric Markov chain converges to the uniform distribution on its state space (e.g. [20]). Its *mixing time* indicates the number of steps necessary to reach the stationary distribution.

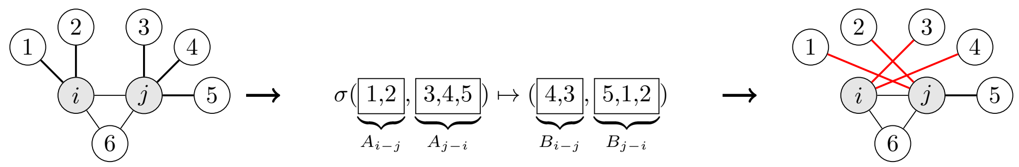
3.1 Edge-Switching

ESMC is a state-of-the-art randomisation method with a wide range of applications, e.g. the generation of graphs [15, 19], or the randomisation of biological datasets [16]. In each step, *ESMC* chooses two edges $e_1 = [u_1, v_1], e_2 = [u_2, v_2]$ and a direction $d \in \{0, 1\}$ uniformly at random and rewires them into $\{u_1, u_2\}, \{v_1, v_2\}$ if d=0 and $\{u_1, v_2\}, \{v_1, u_2\}$ otherwise. If a step yields a non-simple graph, it is skipped. *ESMC*'s Markov chain is irreducible [10], aperiodic and symmetric [23] and hence converges to the uniform distribution on the space of simple graphs with fixed degree sequence. While analytic bounds on the mixing time [12, 13] are impractical, usually a number of steps linear in the number of edges is used in practice [29].

3.2 Simple Undirected Curveball algorithm

Curveball is a novel randomisation method. In each step, two nodes trade their neighbourhoods, possibly yielding faster mixing times [5, 35, 36].

▶ **Definition 1** (Simple Undirected Trade). Let $G = (V, E)$ be a simple graph, A be its adjacency list representation, and A_u be the set of neighbours of node u. A trade $t = (i, j, \sigma)$


$\sigma(\underbrace{\boxed{1,2}}_{A_{i-j}},\underbrace{\boxed{3,4,5}}_{A_{j-i}}) \mapsto (\underbrace{\boxed{4,3}}_{B_{i-j}},\underbrace{\boxed{5,1,2}}_{B_{j-i}})$

$A_i = \{1,2,6,j\}$
$A_j = \{3,4,5,6,i\}$

$B_{i-j} = \{3,4\}$
$B_{j-i} = \{1,2,5\}$

$B_i = \{3,4,6,j\}$
$B_j = \{1,2,5,6,i\}$

Figure 2 The trade (i,j,σ) between nodes i and j only considers edges to the disjoint neighbours $\{1,\dots,5\}$. For the reassigned disjoint neighbours we use the short-hand $B_{i-j} := \{x \mid x \in D_{ij}, \text{rank}_\sigma(x) \le |A_{i-j}|\}$ and $B_{j-i} := \{x \mid x \in D_{ij}, \text{rank}_\sigma(x) > |A_{i-j}|\}$. The triangle $(i,j,6)$ is omitted as trading any of its edges would either introduce parallel edges, self loops, or result in no change at all. Then, the given σ exchanges four edges.

from A to adjacency list B is defined by two nodes i and j, and a permutation $\sigma\colon D_{ij} \to D_{ij}$ where $A_{i-j} := A_i \setminus (A_j \cup \{j\})$ and $D_{ij} := A_{i-j} \cup A_{j-i}$. As shown in Fig. 2, performing t on G results in $B_i = (A_i \backslash A_{i-j}) \cup \{x \mid x \in D_{ij}, \text{rank}_\sigma(x) \le |A_{i-j}|\}$ and $B_j = (A_j \backslash A_{j-i}) \cup \{x \mid x \in D_{ij}, \text{rank}_\sigma(x) > |A_{i-j}|\}$. Since edges are undirected, symmetry has to be preserved: for all $u \in A_i \backslash B_i$ the label j in adjacency list B_u is changed to i and analogously for $A_j \setminus B_j$.

Simple Undirected Curveball randomises a graph by repeatedly selecting a node pair $\{i,j\}$ and permutation σ on the disjoint neighbours uniformly at random. Its Markov chain is irreducible, aperiodic and symmetric and hence converges to the uniform distribution [6].

3.3 Undirected Global Trades

Trade sequences typically consist of pairs in which each constituent is drawn uniformly at random. While it is a well-known fact[3] that $\Theta(n \log n)$ trades are required in expectation until each node is included at least once, there is no apparent reason why this should be beneficial; in fact, experiments in section 5 suggest the contrary.

Carstens et al. propose the notion of *global trades* for directed or bipartite graphs as a 2-partition of all nodes implicitly forming $n/2$ node pairs to be traded in a single step [6]. This concept is not applicable to undirected graphs where in general the two directions (u,v) and (v,u) of an edge $\{u,v\}$ cannot be processed independently in a single step. We hence extend global trades to undirected graphs by interpreting them as a sequence of $n/2$ single trades which together target each node exactly once (we assume n to be even; if this is not the case we add an isolated node). Dependencies are then resolved by the order of this sequence.

▶ **Definition 2** (Undirected Global trade)**.** Let $G = (V,E)$ be a simple graph and $\pi\colon V \to V$ be a permutation on the set of nodes. A *global trade* $T = (t_1,\dots,t_\ell)$ for $\ell = \lfloor n/2 \rfloor$ is a sequence of trades $t_i = \{\pi(v_{2i-1}), \pi(v_{2i}), \sigma_i\}$. By applying T to G we mean that the trades $t_1,\dots,t_\ell$ are applied successively starting with G.

Theorem 3 allows us to use global trades as a substitute for a sequence of single trades, as global trades preserve the stationary distribution of Curveball's Markov chain. The proof extends [6], which shows convergence of global trades in bipartite or directed graphs, to undirected graphs and uses similar techniques.

[3] For instance studied as the coupon collector problem.

▶ **Theorem 3.** *Let $G = (V, E)$ be an arbitrary simple undirected graph, and let Ω_G be the set of all simple directed graphs that have the same degree sequence as G. The Curveball algorithm with global trades and started at G converges to the uniform distribution on Ω_G.*

Proof. In order to prove the claim, we have to show irreducibility and aperiodicity of the Markov chain as well as symmetry of the transition probabilities.

For the first two properties it suffices to show that whenever there exists a single trade from state A to B, there also exists a global trade from A to B (see [4] for a similar argument).[4] Observe that there is a non-zero probability that a single trade does not change the graph, e.g. by selecting σ_i as the identity. Hence there is a non-zero probability that …

- a global trade does not alter the graph at all. This corresponds to a self-loop at each state of the Markov chain and hence guarantees aperiodicity.
- all but one single trade of a global trade do not alter the graph. In this case, a global trade degenerates to a single trade and the irreducibility shown in [4] carries over.

It remains to show that the transition probabilities are symmetric. Let $\mathcal{T}^g_{AB}$ be the set of global trades that transform state A to state B. Then the transition probability between A and B equals the sum of probabilities of selecting a trade sequence from $\mathcal{T}^g_{AB}$. That is $P_{AB} = \sum_{T \in \mathcal{T}^g_{AB}} \mathbf{P}_A(T)$ where $\mathbf{P}_A(T)$ denotes the probability of selecting global trade T in state A.

The probability $\mathbf{P}_A(t)$ of selecting a single trade $t = (i, j, \sigma)$ from state A to state B equals the probability $\mathbf{P}_B(\tilde{t})$ of selecting the reverse trade $\tilde{t} = (i, j, \sigma^{-1})$ from state B to A [6]. We now define the reverse global trade of $T = (t_1, \ldots, t_\ell)$ as $\tilde{T} = (\tilde{t}_\ell, \ldots, \tilde{t}_1)$. It is straight-forward to check that this gives a bijection between the sets $\mathcal{T}^g_{AB}$ and $\mathcal{T}^g_{BA}$.

It remains to show that the middle equality holds in

$$P_{AB} = \sum\nolimits_{T \in \mathcal{T}^g_{AB}} \mathbf{P}_A(T) \overset{!}{=} \sum\nolimits_{\tilde{T} \in \mathcal{T}^g_{BA}} \mathbf{P}_B(\tilde{T}) = P_{BA}.$$

Let $T = (t_1, \ldots, t_\ell)$ be a global trade from A to B as implied by π and $A{=}A_1, \ldots, A_{\ell+1}{=}B$ be the intermediate states. We denote the reversal of T and π as $\tilde{T}$ and $\tilde{\pi}$ respectively and obtain $P_A(T) = \mathbf{P}(\pi)\mathbf{P}_{A_1}(t_1) \ldots \mathbf{P}_{A_\ell}(t_\ell) = \mathbf{P}(\tilde{\pi})\mathbf{P}_B(\tilde{t}_\ell) \ldots \mathbf{P}_{A_2}(\tilde{t}_1) = P_B(\tilde{T})$. Clearly $\mathbf{P}(\pi) = \mathbf{P}(\tilde{\pi})$ as we are picking permutations uniformly at random. The second equality follows from $\mathbf{P}_A(t) = \mathbf{P}_B(\tilde{t})$ for a single trade between A and B. ◀

4 Novel Curveball algorithms for undirected graphs

In this section we present the related algorithms *EM-CB*, *IM-CB*, *EM-GCB* and *EM-PGCB*. The algorithms receive a simple graph G and a *trade sequence* $T = [\,\{u_i, v_i\}\,]_{i=1}^{\ell}$ as input and compute the result of carrying out the trade sequence T (see section 3.2) in order.

EM-CB and *IM-CB* are sequential solutions suited to process arbitrary trade sequences T. For our analysis, we assume T's constituents to be drawn uniformly at random (as expected in typical applications). Both algorithms share a common design, but differ in the data structures used. *EM-CB* is an I/O-efficient algorithm while *IM-CB* is optimised for small graphs allowing for unstructured accesses to main memory. In contrast, *EM-GCB* and *EM-PGCB* process global trades only. This restricted input model allows us to represent the trade sequence T implicitly by hash functions which further accelerates trading.

[4] Since each global trade can be emulated by its $n/2$ decomposed single trades, the reverse is true for a hop of $n/2$ single trade steps. Due to dependencies however the transition probabilities generally do not match, see $V = \{1, 2, 3, 4\}$ and $E = \{[1, 2], [3, 4]\}$ for a simple counterexample.

Algorithm 1: *EM-CB*.

```
Data: Trade sequence $T$, simple graph $G = (V, E)$ by edge list $E$
// Preprocessing: Compute Dependencies
1  foreach trade $t_i = (u, v) \in T$ for increasing $i$ do
2      Send messages $\langle u, t_i\rangle$ and $\langle v, t_i\rangle$ to Sorter SorterTtoV
3  Sort SorterTtoV lexicographically        // All trades of a node are next to each other
4  foreach node $u \in V$ do
5      Receive $\mathcal{S}(u) = [t_1, \ldots, t_k]$ from $k$ messages addressed to $u$ in SorterTtoV
6      Set $t_{k+1} \leftarrow \infty$                    // $t_1 = \infty$ iff $u$ is never active
7      Send $\langle t_i, u, t_{i+1}\rangle$ to SorterDepChain for $i \in [k]$
8      foreach directed edge $(u, v) \in E$ do
9          if $u < v$ then
10             Send message $\langle v, u, t_1\rangle$ via PqVtoV
11         else
12             Receive $t_1^v$ from unique message received via PqVtoV
13             if $t_1 \leq t_1^v$ then Send message $\langle t_1, u, v, t_1^v\rangle$ via PqTtoT
               else                 Send message $\langle t_1^v, v, u, t_1\rangle$ via PqTtoT
14 Sort SorterDepChain
// Main phase - Currently at least the first trade has all information it needs
15 foreach trade $t_i = (u, v) \in T$ for increasing $i$ do
16     Receive successors $\tau(u)$ and $\tau(v)$ via SorterDepChain
17     Receive neighbours $\mathcal{A}_G(u)$, $\mathcal{A}_G(v)$ and their successors $\tau(\cdot)$ from PqTtoT
18     Randomly reassign disjoint neighbours, yielding new neighbours $\mathcal{A}'_G(u)$ and $\mathcal{A}'_G(v)$.
19     foreach $(a, b) \in (\{u\} \times \mathcal{A}'_G(u)) \cup (\{v\} \times \mathcal{A}'_G(v))$ do
           if $\tau_a = \infty$ and $\tau_b = \infty$ then Output final edge $\{a, b\}$
20         else if $\tau_a \leq \tau_b$ then      Send message $\langle \tau_a, a, b, \tau_b\rangle$ via PqTtoT
           else                              Send message $\langle \tau_b, b, a, \tau_a\rangle$ via PqTtoT
```

At core, all algorithms perform trades in a similar fashion: In order to carry out the i-th trade $\{u_i, v_i\}$, they retrieve the neighbourhoods $\mathcal{A}_{u_i}$ and $\mathcal{A}_{v_i}$, shuffle[5] them, and then update the graph. Once the neighbourhoods are known, trading itself is straight-forward. We compute the set of disjoint neighbours $D = (\mathcal{A}_{u_i} \cup \mathcal{A}_{v_i}) \setminus (\mathcal{A}_{u_i} \cap \mathcal{A}_{v_i})$ and then draw $|\mathcal{A}_{u_i} \cap D|$ nodes from D for u_i uniformly at random while the remaining nodes go to v_i. If $\mathcal{A}_{u_i}$ and $\mathcal{A}_{v_i}$ are sorted this requires only $\mathcal{O}(|\mathcal{A}_{u_i}| + |\mathcal{A}_{v_i}|)$ work and scan$(|\mathcal{A}_{u_i}| + |\mathcal{A}_{v_i}|)$ I/Os (see also proof of Lemma 6 if the neighbourhoods fit into RAM). Hence we focus on the harder task of obtaining and updating the adjacency information.

4.1 EM-CB: A sequential I/O-efficient Curveball algorithm

EM-CB is an I/O-efficient Curveball algorithm to randomise undirected graphs as detailed in Alg. 1. This basic algorithm already contains crucial design principles which we further explore with *IM-CB*, *EM-GCB* and *EM-PGCB* in sections 4.2 and 4.4 respectively.

The algorithm encounters the following challenges. After an undirected trade $\{u, v\}$ is carried out, it does not suffice to only update the neighbourhoods $\mathcal{A}_u$ and $\mathcal{A}_v$: consider the case that edge $\{u, x\}$ changes into $\{v, x\}$. Then this switch also has to be reflected in the neighbourhood of $\mathcal{A}_x$. Here, we call u and v *active* nodes while x is a *passive* neighbour.

[5] In contrast to Definition 2, we do not consider the permutation σ of disjoint neighbours as part of the input, but let the algorithm choose one randomly for each trade. We consider this design decision plausible as the set of disjoint neighbours only emerges over the course of the execution.

In the EM setting another challenge arises for graphs exceeding main memory; it is prohibitively expensive to directly access the edge list since this unstructured pattern triggers $\Omega(1)$ I/Os for each edge processed with high probability.

EM-CB approaches these issues by abandoning a classical static graph data structure containing two redundant copies of each edge. Following the *TFP* principle, we rather interpret all trades as a sequence of points over time that are able to receive messages. Initially, we send each edge to the earliest trade one of its endpoints is active in.[6] This way, the first trade receives one message from each neighbour of the active nodes and hence can reconstruct $\mathcal{A}_{u_1}$ and $\mathcal{A}_{v_1}$. After shuffling and reassigning the disjoint neighbours, *EM-CB* sends each resulting edge to the trade which requires it next. If no such trade exists, the edge can be finalised by committing it to the output.

The algorithm hence requires for each (actively or passively) traded node u, the index of the next trade in which u is actively processed. We call this the *successor* of u and define it to be ∞ if no such trade exists. The dependency information is obtained in a preprocessing step; given $T = [\,\{u_i, v_i\}\,]_{i=1}^{\ell}$, we first compute for each node u the monotonically increasing index list $\mathcal{S}(u)$ of trades in which u is actively processed, i.e. $\mathcal{S}(u) := \left[\, i \mid u \in t_i \text{ for } i \in [\ell] \,\right] \circ [\infty]$.

▶ **Example 4.** Let $G = (V, E)$ be a simple graph with $V = \{v_1, v_2, v_3, v_4\}$ and trade sequence $T = [t_1\colon \{v_1, v_2\}, t_2\colon \{v_3, v_4\}, t_3\colon \{v_1, v_3\}, t_4\colon \{v_2, v_4\}, t_5\colon \{v_1, v_4\}]$. Then, the successors $\mathcal{S}$ follow as $\mathcal{S}(v_1) = [1, 3, 5, \infty]$, $\mathcal{S}(v_2) = [1, 4, \infty]$, $\mathcal{S}(v_3) = [2, 3, \infty]$, $\mathcal{S}(v_4) = [2, 4, 5, \infty]$.

This information is then spread via two channels:

- After preprocessing, *EM-CB* scans $\mathcal{S}$ and T conjointly and sends $\langle t_i, u_i, t_i^u \rangle$ and $\langle t_i, v_i, t_i^v \rangle$ to each trade t_i. The messages carry the successors t_i^u and t_i^v of the trade's active nodes.
- When sending an edge as described before, we augment it with the successor of the passive node. Initially, this information is obtained by scanning the edge list E and $\mathcal{S}$ conjointly. Later, it can be inductively computed since each trade receives the successors of all nodes involved.

▶ **Lemma 5.** *For an arbitrary trade sequence T of length ℓ, EM-CB has a worst-case I/O complexity of $\mathcal{O}[\mathrm{sort}(\ell) + \mathrm{sort}(n) + \mathrm{scan}(m) + \ell d_{\max}/B \log_{M/B}(m/B)]$. For r global trades, the worst case I/O complexity is $\mathcal{O}(r[\mathrm{sort}(n) + \mathrm{sort}(m)])$.*

Proof. Refer to the full article [7] for the proof. ◀

4.2 IM-CB: An internal memory version of EM-CB

While *EM-CB* is well-suited if memory access is a bottleneck, we also consider the modified version *IM-CB*. As shown in section 5, *IM-CB* is typically faster for small graph instances. *IM-CB* uses the same algorithmic ideas as *EM-CB* but replaces its priority queues and sorters[7] by unstructured I/O into main memory (see [7] for details):

- Instead of sending neighbourhood information in a TFP-fashion, we now rely on a classical adjacency vector data structure $\mathcal{A}_G$ (an array of arrays). Similarly to *EM-CB*, we only

[6] If an edge connects two nodes that are both actively traded we implicitly perform an arbitrary tie-break.

[7] The term *sorter* refers to a container with two modes of operation: in the first phase, items are pushed into the write-only sorter in an arbitrary order by some algorithm. After an explicit switch, the filled data structure becomes read-only and the elements are provided as a lexicographically non-decreasing stream which can be rewound at any time. While a sorter is functionally equivalent to filling, sorting and reading back an EM vector, the restricted access model reduces constant factors in the implementation's runtime and I/O-complexity [3].

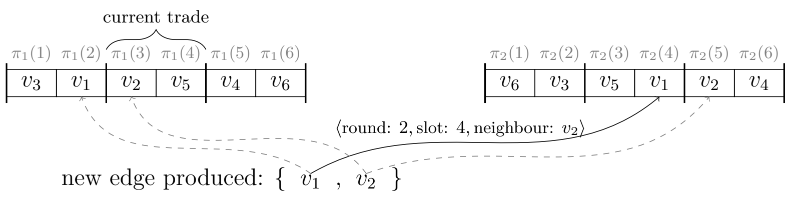

current trade
$\pi_1(1)$ $\pi_1(2)$ $\pi_1(3)$ $\pi_1(4)$ $\pi_1(5)$ $\pi_1(6)$
v_3 v_1 v_2 v_5 v_4 v_6
$\pi_2(1)$ $\pi_2(2)$ $\pi_2(3)$ $\pi_2(4)$ $\pi_2(5)$ $\pi_2(6)$
v_6 v_3 v_5 v_1 v_2 v_4
⟨round: 2, slot: 4, neighbour: v_2⟩
new edge produced: { v_1 , v_2 }

Figure 3 During the trade j=1, i_1=3, i_2=4 the edge $\{v_1, v_2\}$ is produced; the arrows indicate positions considered as successors. Since v_1 and v_2 are already processed in round j=1, π_2 is used to compute the successor. Then, the message is sent to v_1 in round 2 as v_1 is processed before v_2.

keep one directed representation of an undirected edge. As an invariant, an edge is always placed in the neighbourhood of the incident node traded before the other. To speed-up these insertions, *IM-CB* maintains unordered neighbourhood buffers.

- *IM-CB* does not forward successor information, but rather stores $\mathcal{S}$ in a contiguous block of memory. The algorithm additionally maintains the vector $\mathcal{S}_{\text{idx}}[1 \dots n]$ where the i-th entry points to the current successor of node v_i. Once this trade is reached, the pointer is incremented giving the next successor.

▶ **Lemma 6.** *For a random trade sequence T of length ℓ, IM-CB has an expected running time of $\mathcal{O}(n + \ell + m + \ell m/n)$. In the case of r many global trades (each consisting of $n/2$ normal trades) the running time is given by $\mathcal{O}(n + rm)$.*

Proof. Refer to the full article [7] for the proof. ◀

4.3 EM-GCB: An I/O-efficient Global Curveball algorithm

EM-GCB builds on *EM-CB* and exploits the regular structure of global trades to simplify and accelerate the dependency tracking. As discussed in section 3.3, a global trade can be encoded as a permutation $\pi\colon [n] \to [n]$ by interpreting adjacent ranks as trade pairs, i.e. $T_\pi = [\{v_{\pi(2i-1)}, v_{\pi(2i)}\}]_{i=1}^{n/2}$. In this setting, a sequence of global trades is given by r permutations $[\pi_j]_{j=1}^{r}$. The model simplifies dependencies as it is not necessary to explicitly gather $\mathcal{S}$ and communicate successors.

As illustrated in Fig. 3, we also change the addressing scheme of messages. While *EM-CB* sends messages to specific nodes in specific trades, *EM-GCB* exploits that each node v_i is actively traded only once in each round j and hence can be addressed by its position $\pi_j(i)$. Successors can then be computed in an ad hoc fashion; let a trade of adjacent positions $i_1 < i_2$ of the j-th global trade produce (amongst others) the edge $\{v_x, v_y\}$. The successor of v_x (and analogously the one of v_y) is $\mathcal{S}_{j,i_2}[v_x] = (j, \pi_j(x))$ if v_x is processed later in round j (i.e. $\pi_j(x)/2 > i_2$) and otherwise $\mathcal{S}_{j,i_2}[v_x] = (j{+}1, \pi_{j+1}(x))$. Here we imply an untraded additional function $\pi_{r+1}(x) = x$ which avoids corner cases and generates an ordered edge list as a result of the r-th global trade.

To reduce the computational cost of the successor computation, *EM-GCB* supports fast injective functions $f\colon X \to Y$ where $[n] \subseteq X$ and $[n] \subseteq Y$. In contrast to the original permutations, their relevant image $\{ f(x) \mid x \in [n] \}$ may contain gaps which are simply skipped by *EM-GCB*. This requires minor changes in the addressing scheme.

In practice, we use functions from the family of linear congruential maps $H_p := \{ h_{a,b} \mid 1 \leq a < p \text{ and } 0 \leq b < p \}$ with $h_{a,b}(x) \equiv [(ax + b) \mod p]$ where p is the smallest prime number $p \geq n$. Random choices from H_p are well suited for *EM-GCB* since they

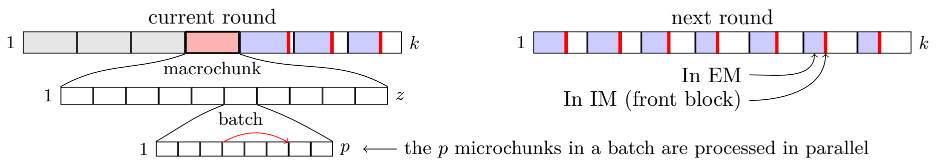


Figure 4 *EM-PGCB* splits each global trade into k *macrochunks* and maintains an external memory queue for each. Before processing a macrochunk, the buffer is loaded into IM and sorted, and further subdivided into z batches each consisting of p microchunks. A type (ii) message is visualised by the red intra-batch arrow.

are 2-universal[8] and contain only $\mathcal{O}(\log(n))$ gaps (see [7] for details). They are also bijections with an easily computable inverse $h^{-1}_{a,b}$ that allows *EM-GCB* to determine the active node $h^{-1}_{a,b}(i)$ traded at position i; this operation is only performed once for each traded position. *EM-GCB* also supports non-invertible functions. This can be implemented with messages $\langle h(i), i\rangle$ that are generated for $1 \le i \le n$ and delivered using TFP.

4.4 EM-PGCB: An I/O-efficient parallel Global Curveball algorithm

EM-PGCB adds parallelism to *EM-GCB* by concurrently executing multiple sequential trades. As in Fig. 4, we split a global trade into *microchunks* each containing a similar number of node pairs and then execute a *batch* of p such subdivisions in parallel. The batch's size is a compromise between intra-batch dependencies (messages are awaited from another processor) and overhead caused by synchronising threads at the batch's end (see [7] for details).

EM-PGCB processes each microchunk similarly as in *EM-CB* but differentiates between messages that are sent (i) within a microchunk, (ii) between microchunks of the same batch (iii) and microchunks processed later. Each class is transported using an optimised data structure (see below) and only type (ii) messages introduce dependencies between parallel executions and are resolved as follows: each processor retrieves the messages that are sent to its next trade and checks whether all information required is available by comparing the number of messages to the active nodes' degrees. If data is missing the trade is skipped and later executed by the processor that adds the last missing neighbour.

For graphs with $m = \mathcal{O}(M^2/B)$ edges[9], we optimise the communication structure for type (iii) messages. Observe that *EM-PGCB* sends messages only to the current and the subsequent round. We partition a round into k *macrochunks* each consisting of $\Theta(n/k)$ contiguous trades. An external memory queue is used for each macrochunk to buffer messages sent to it; in total, this requires $\Theta(kB)$ internal memory. Before processing a macrochunk, all its messages are loaded into IM, subsequently sorted and arranged such that missing messages can be directly placed to the position they are required in. This can also be overlapped with the processing of the previous macrochunk. The number k of macrochunks should be as small as possible to reduce overheads, but sufficiently large such that all messages of a macrochunk fit into main memory (see [7] for details).

▶ **Theorem 7.** *EM-PGCB requires $\mathcal{O}(r \cdot [\text{sort}(n) + \text{sort}(m)])$ I/Os to perform r global trades.*

Proof. Observe that we can analyse each of the r rounds individually. A constant amount of auxiliary data is needed per node to provision gaps for missing data, to detect whether a

[8] i.e. given one node in a single trade, the other is uniformly chosen among the remaining nodes.
[9] Even with as little as 1 GiB of internal memory, several billion edges are supported.

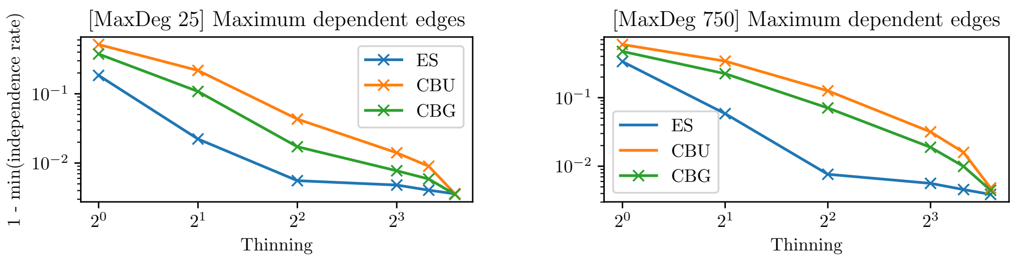


Figure 5 Fraction of edges still correlated as a function of the thinning parameter k for graphs with $n = 2{\cdot}10^3$ nodes and degree distribution $\textsc{Pld}([a, b), \gamma)$ with $\gamma = 2$, $a = 5$, and $b \in \{25, 750\}$. The (not thinned) long Markov chains of edge switching (ES), uniform Curveball (CBU) and global Curveball (CBG) contain 6000 super steps each.

trade can be executed and (if required) to invert the permutation. This accounts for $\Theta(n)$ messages requiring sort(n) I/Os to be delivered. Using an ordinary PQ, the analysis of *EM-CB* (see Lemma 5) carries over, requiring sort(m) I/Os for a global trade. ◀

5 Experimental Evaluation

In this section we evaluate the quality of the proposed algorithms and analyse the runtime of our C++ implementations.[10] *EM-CB*, *IM-CB*, *EM-GCB* are designed as modules of NetworKit [33]; due to their superior performance, only the latter two were added to the library and are available since release 4.6. *EM-PGCB*'s implementation is developed separately and facilitates external memory data structures and algorithms of STXXL [9].

Intuitively, graphs with skewed degree distributions are hard instances for Curveball since it shuffles and reassigns the disjoint neighbours of two trading nodes. Hence, limited progress is achieved if a high-degree node trades with a low-degree node. Since our experiments support this hypothesis, we focus on graphs with powerlaw degree distributions as difficult but highly relevant graph instances. Our experiments use two parameter sets:

- *(lin)* – The maximal possible degree scales linearly as a function of the number n of nodes. The degree distribution $\textsc{Pld}([a, b), \gamma)$ is chosen as $a = 10$, $b = n/20$ and $\gamma = 2$.
- *(const)* – The extremal degrees are kept constant. In this case the parameters are chosen as $a = 50$, $b = 10000$ and $\gamma = 2$.

We select these configurations to be comparable with [15] where both parameter sets are used to evaluate *EM-ES*. The first setting *(lin)* considers the increasing average degree of real-world networks as they grow. The second setting *(const)* approximates the degree distribution of the Facebook network in May 2011 (refer to [14] for details). Runtimes are measured on the following off-the-shelf machine: Intel Xeon E5-2630 v3 (8 cores at 2.40GHz), 64GB RAM, 2× Samsung 850 PRO SATA SSD (1 TB), Ubuntu Linux 16.04, GCC 7.2.

5.1 Mixing of Edge-Switching, Curveball and Global Curveball

We are not aware of any practical theoretical bounds on the mixing time of Markov chains of Curveball, Global Curveball or edge switching (see section 3). Hence, we quantitatively study the progress made by Curveball trades compared to edge switching and approximate the

[10] Code used for the presented benchmarks can be found at our fork `https://github.com/hthetran/networkit` (*IM-CB* and *EM-CB*) and `https://github.com/massive-graphs/extmem-lfr` (*EM-PGCB*).

mixing time of the underlying Markov chains by a method developed in [30]. This criterion is a more sensitive proxy to the mixing time than previously used alternatives, such as the local clustering coefficient, triangle count and degree assortativity [14].

Intuitively, one determines the number of Markov chain steps required until the correlation to the initial state decays. Starting from an initial graph G_0, the Markov chain is executed for a large number of steps, yielding a sequence $(G_t)_{t \geq 0}$ of graphs evolving over time. For each occurring edge e, we compute a boolean vector $(Z_{e,t})_{t \geq 0}$ where a 1 at position t indicates that e exists in graph G_t. We then derive the *k-thinned* series $(Z^k_{e,t})_{t \geq 0}$ only containing every k-th entry of the original vector $(Z_{e,t})_{t \geq 0}$ and use k as a proxy for the mixing time.

To determine if k Markov chain steps suffice for edge e to lose the correlation to the initial graph, the empirical transition probabilities of the k-thinned series $(Z^k_{e,t})_{t \geq 0}$ are fitted to both an independent and a Markov model respectively. If the independent model is a better fit, we deem edge e to be independent.

The results presented here consider only small graphs due to the high computational cost involved. However, additional experiments suggest that the results hold for graphs at least one order of magnitude larger which is expected as powerlaw distributions are scale-free.

We compare a sequence of uniform (single) trades, global trades and edge switching and visually align the results of these schemes by defining a *super step*. Depending on the algorithm a super step corresponds to either a single global trade, $n/2$ uniform trades or m edge-swaps. Comparing $n/2$ uniform trades with a global trade seems sensible since a global trade consists of exactly $n/2$ single trades, furthermore randomising with $n/2$ single trades considers the state of $2m$ edges which is also true for m edge-swaps. The alignment accounts for the fact that a single Curveball Markov chain step may execute multiple neighbour switches, thus easily outperforming *ESMC* in a step-by-step comparison.

Fig. 5 contains a selection of results obtained for small powerlaw graph instances using this method (see [7] for the complete dataset). Progress is measured by the fraction of edges that are still classified as correlated, i.e. the faster a method approaches zero the better the randomisation. We omit an in-depth discussion of uniform trades and rather focus on global trades which consistently outperform the former (cf. section 3.2).

In all settings *ESMC* shows the fastest decay. The gap towards global trades growths temporarily as the maximal degree is increased which is consistent with our initial claim that skewed degree distributions are challenging for Curveball. The effect is however limited and in all cases performing 4 global trades for each edge switching super step gives better results. This is a pessimistic interpretation since typically $10m$ to $100m$ edge switches are used to randomise graphs in practice; in this domain global trades perform similarly well and 20 global trades consistently give at least the quality of $10m$ edge switches.

5.2 Runtime performance benchmarks

We measure the runtime of the algorithms proposed in section 4 and compare them to two state-of-the-art edge switching schemes (using the authors' C++ implementations):

- *VL-ES* is a sequential IM algorithm with a hashing-based data structure optimised for efficient neighbourhood queries and updates [37]. To achieve comparability, we removed connectivity tests, fixed memory management issues, and adopted the number of swaps.
- *EM-ES* is an EM edge switching algorithm and part of *EM-LFR*'s toolchain [15].

We carry out experiments using the *(const)* and *(lin)* parameter sets, and limit the problem sizes for internal memory algorithms to avoid exhaustion of the main memory. For

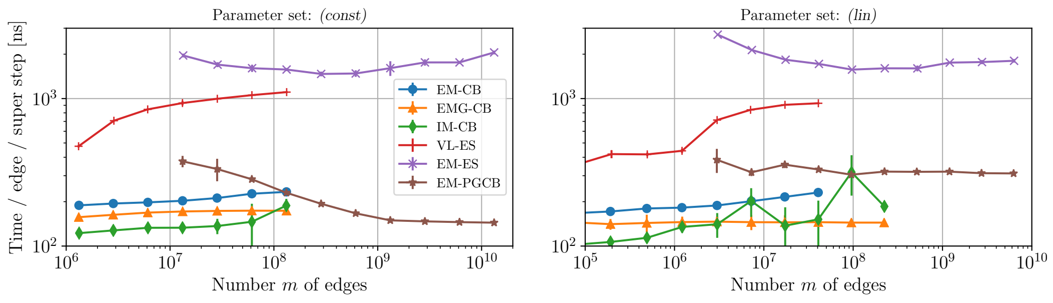


Figure 6 Runtime per edge and super step (global trade or m edge swaps) of the proposed algorithms *IM-CB*, *EM-CB* and *EM-PGCB* compared to state-of-the-art IM edge switching *VL-ES* and EM edge switching *EM-ES*. Each data point is the median of $S \geq 5$ runs over 10 super steps each. The left plot contains the (const)-parameter set, the right one (lin). Observe that the super steps of different algorithms advance the randomisation process at different speeds (see discussion).

each data point we carry out 10 super steps (i.e. 10 global trades or $10m$ edge swaps) on a graph generated with Havel-Hakimi from a random powerlaw degree distribution.

Figure 6 presents the walltime per edge and super step including pre-computation[11] required by the algorithms but excluding the initial graph generation process. The plots include (mostly small) errorbars corresponding to the unbiased estimation of the standard deviation of S repetitions per data point (with different random seeds).

The number k of macrochunks does not significantly affect *EM-PGCB*'s performance for small graphs due to comparably high synchronisation cost. In contrast, adjusting k for larger graphs can noticeably increase the performance of *EM-PGCB*. We thus experimentally determined the value $k = 32$ for both *(const)* and *(lin)* with $n = 10^7$ nodes and use that value for all other instances.

All Curveball algorithms outperform their direct competitors significantly – even if we pessimistically executed two global trades for each edge switching super step (see section 5.1). For large instances of *(const)* *EM-PGCB* carries out one super step 14.3 times faster than *EM-ES* and 5.8 times faster for *(lin)*. *EM-PGCB* also shows a superior scaling behaviour with an increasing speed-up for larger graphs. Similarly, *IM-CB* processes super steps up to 6.3 times faster than *VL-ES* on *(const)* and 5.1 times on *(lin)*.

On our test machine, the implementation of *IM-CB* outperforms *EM-CB* in the internal memory regime; *EM-GCB* is faster for large graphs. As indicated in [7], this changes qualitatively for machines with slower main memory and smaller cache; on such systems the unstructured I/O of *IM-CB* and *VL-ES* is more significant rendering *EM-CB* and *EM-GCB* the better choice with a speed-up factor exceeding 8 compared to *VL-ES*.

6 Conclusion and outlook

We applied *global* Curveball trades to undirected graphs simplifying the algorithmic treatment of dependencies and showed that the underlying Markov chain converges to a uniform distribution. Experimental results show that global trades yield an improved quality compared to a sequence of uniform trades of the same size.

[11] For *VL-ES* we report only the swapping process and the generation of the internal data structures.

We presented *IM-CB* and *EM-CB*, the first efficient algorithms for Simple Undirected Curveball algorithms; they are optimised for internal and external memory respectively. Our I/O-efficient parallel algorithm *EM-PGCB* exploits the properties of global trades and executes a super step 14.3 times faster than the state-of-the-art edge switching algorithm *EM-ES*; for *IM-CB* we demonstrate speed-ups of up to 6.3 (in a conservative comparison the speed-ups should be halved to account for the differences in mixing times of the underlying Markov chains). The implementations of all three algorithms are freely available and are in the process of being incorporated into *EM-LFR* and considered for NetworKit.

References

1 A. Aggarwal, J. Vitter, et al. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988. doi:10.1145/48529.48535.

2 L. Arge. *The buffer tree: A new technique for optimal I/O-algorithms*, pages 334–345. Springer Berlin Heidelberg, 1995. doi:10.1007/3-540-60220-8_74.

3 A. Beckmann, R. Dementiev, and J. Singler. Building a parallel pipelined external memory algorithm library. In *IPDPS'09*, 2009. doi:10.1109/IPDPS.2009.5161001.

4 C. J. Carstens. Proof of uniform sampling of binary matrices with fixed row sums and column sums for the fast curveball algorithm. *Physical Review E*, 91:042812, 2015.

5 C. J. Carstens. *Topology of Complex Networks: Models and Analysis*. PhD thesis, RMIT University, January 2016.

6 C. J. Carstens, A. Berger, and G. Strona. Curveball: a new generation of sampling algorithms for graphs with fixed degree sequence. *CoRR*, 2016. arXiv:1609.05137.

7 C. J. Carstens, M. Hamann, U. Meyer, M. Penschuck, H. Tran, and D. Wagner. Parallel and I/O-efficient randomisation of massive networks using global curveball trades. *CoRR*, abs/1804.08487, 2018.

8 G. W. Cobb and Y.-P. Chen. An application of markov chain monte carlo to community ecology. *The American Mathematical Monthly*, 110(4):265–288, 2003.

9 R. Dementiev, L. Kettner, and P. Sanders. STXXL: standard template library for XXL data sets. *Software: Practice and Experience*, 38(6):589–637, 2008. doi:10.1002/spe.844.

10 R. B. Eggleton and D. A. Holton. *Simple and multigraphic realizations of degree sequences*, pages 155–172. Springer Berlin Heidelberg, 1981. doi:10.1007/BFb0091817.

11 P. Erdős and A. Rényi. On random graphs I. *Publicationes Mathematicae Debrecen*, 1959.

12 C. Greenhill. A polynomial bound on the mixing time of a markov chain for sampling regular directed graphs. *The Electronic Journal of Combinatorics*, 18(1):P234, 2011.

13 C. Greenhill. The switch markov chain for sampling irregular graphs: Extended abstract. In *Proceedings of SODA '15*, pages 1564–1572, 2015.

14 M. Hamann, U. Meyer, M. Penschuck, H. Tran, and D. Wagner. I/O-efficient generation of massive graphs following the LFR benchmark. *CoRR*, 2017. arXiv:1604.08738.

15 M. Hamann, U. Meyer, M. Penschuck, and D. Wagner. I/O-efficient generation of massive graphs following the LFR benchmark. In *ALENEX*, 2017. doi:10.1137/1.9781611974768.

16 F. Iorio, M. Bernardo-Faura, A. Gobbi, T. Cokelaer, G. Jurman, and J. Saez-Rodriguez. Efficient randomization of biological networks while preserving functional characterization of individual nodes. *BMC bioinformatics*, 17(1):542, 2016.

17 S. Itzkovitz, R. Milo, N. Kashtan, G. Ziv, and U. Alon. Subgraphs in random networks. *Physical review E*, 68:026127, Aug 2003. doi:10.1103/PhysRevE.68.026127.

18 A. Lancichinetti and S. Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Phys. Rev. E*, 80:016118, Jul 2009. doi:10.1103/PhysRevE.80.016118.

19 A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Phys. Rev. E*, 78:046110, 2008. `doi:10.1103/PhysRevE.78.046110`.

20 D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov chains and mixing times.* American Mathematical Society, Providence, Rhode Island, 2009.

21 A. Maheshwari and N. Zeh. *A Survey of Techniques for Designing I/O-Efficient Algorithms*, pages 36–61. Springer Berlin Heidelberg, 2003.

22 U. Meyer, P. Sanders, and J. Sibeyn. *Algorithms for Memory Hierarchies: Advanced Lectures.* Springer Berlin Heidelberg, 2003. `doi:10.1007/3-540-36574-5`.

23 C. G. M. Mihail and E. Zegura. The markov chain simulation method for generating connected power law random graphs. In *Proceedings of ALENEX '03.* SIAM, 2003.

24 R. Milo, N. Kashtan, S. Itzkovitz, M. E. J. Newman, and U. Alon. On the uniform generation of random graphs with prescribed degree sequences. *CoRR*, 2003. `arXiv:cond-mat/0312028`.

25 M. Molloy and B. Reed. A critical point for random graphs with a given degree sequence. *Random Struct. Algorithms*, 6(2/3):161–179, 1995.

26 M. E. J. Newman. The Structure and Function of Complex Networks. *SIAM Review*, 45(2):167–256, 2003. `doi:10.1137/S003614450342480`.

27 M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E*, 64:026118, Jul 2001. `doi:10.1103/PhysRevE.64.026118`.

28 R. Pagh. *Basic external memory data structures*, pages 36–61. Springer Berlin Heidelberg, 2003.

29 J. Ray, A. Pinar, and C. Seshadhri. *Are We There Yet? When to Stop a Markov Chain while Generating Random Graphs*, pages 153–164. Springer Berlin Heidelberg, 2012. `doi:10.1007/978-3-642-30541-2_12`.

30 J. Ray, A. Pinar, and C. Seshadhri. A stopping criterion for markov chains when generating independent random graphs. *J. of Compl. Net.*, 3(2), 2015. `doi:10.1093/comnet/cnu041`.

31 W. E. Schlauch, E. Á. Horvát, and K. A. Zweig. Different flavors of randomness: comparing random graph models with fixed degree sequences. *Social Network Analysis and Mining*, 5(1):1–14, 2015. `doi:10.1007/s13278-015-0267-z`.

32 W. E. Schlauch and K. A. Zweig. Influence of the null-model on motif detection. In *ASONAM'15*, NY, USA, 2015. ACM. `doi:10.1145/2808797.2809400`.

33 C. L. Staudt, A. Sazonovs, and H. Meyerhenke. NetworKit: A tool suite for large-scale complex network analysis. *Network Science*, 4(04), 2016. `doi:10.1017/nws.2016.20`.

34 S. H. Strogatz. Exploring complex networks. *Nature*, 410(6825):268, 2001.

35 G. Strona, D. Nappo, F. Boccacci, S. Fattorini, and J. San-Miguel-Ayanz. A fast and unbiased procedure to randomize ecological binary matrices with fixed row and column totals. *Nature Communications*, 5:4114–, 2014. `doi:10.1038/ncomms5114`.

36 N. D. Verhelst. An efficient MCMC algorithm to sample binary matrices with fixed marginals. *Psychometrika*, 73(4):705–728, 2008.

37 F. Viger and M. Latapy. Fast generation of random connected graphs with prescribed degrees. *CoRR*, feb 2005. Source code available at `https://www-complexnetworks.lip6.fr/~latapy/FV/generation.html`. `arXiv:cs/0502085`.

Revision Notice

This is a revised version of the eponymous paper appeared in the proceedings of ESA 2018 (LIPIcs, volume 112, `http://www.dagstuhl.de/dagpub/978-3-95977-081-1` published in August, 2018), in which the concept of global trades is now correctly attributed to Carstens, Berger, Strona. Curveball: a new generation of sampling algorithms for graphs with fixed degree sequence. arXiv:1609.05137.

Dagstuhl Publishing – August 27, 2018.

Space-Optimal Quasi-Gray Codes with Logarithmic Read Complexity

Diptarka Chakraborty
Computer Science Institute of Charles University, Prague, Czech Republic
diptarka@iuuk.mff.cuni.cz

Debarati Das
Computer Science Institute of Charles University, Prague, Czech Republic
debaratix710@gmail.com

Michal Koucký
Computer Science Institute of Charles University, Prague, Czech Republic
koucky@iuuk.mff.cuni.cz

Nitin Saurabh
Max-Planck-Institut für Informatik, Saarbrücken, Germany
nsaurabh@mpi-inf.mpg.de

Abstract

A quasi-Gray code of dimension n and length ℓ over an alphabet Σ is a sequence of distinct words $w_1, w_2, \dots, w_\ell$ from Σ^n such that any two consecutive words differ in at most c coordinates, for some fixed constant $c > 0$. In this paper we are interested in the read and write complexity of quasi-Gray codes in the bit-probe model, where we measure the number of symbols read and written in order to transform any word w_i into its successor w_{i+1}.

We present construction of quasi-Gray codes of dimension n and length 3^n over the ternary alphabet $\{0, 1, 2\}$ with worst-case read complexity $O(\log n)$ and write complexity 2. This generalizes to arbitrary odd-size alphabets. For the binary alphabet, we present quasi-Gray codes of dimension n and length at least $2^n - 20n$ with worst-case read complexity $6 + \log n$ and write complexity 2. This complements a recent result by Raskin [Raskin '17] who shows that any quasi-Gray code over binary alphabet of length 2^n has read complexity $\Omega(n)$.

Our results significantly improve on previously known constructions and for the odd-size alphabets we break the $\Omega(n)$ worst-case barrier for space-optimal (non-redundant) quasi-Gray codes with constant number of writes. We obtain our results via a novel application of algebraic tools together with the principles of catalytic computation [Buhrman et al. '14, Ben-Or and Cleve '92, Barrington '89, Coppersmith and Grossman '75].

2012 ACM Subject Classification Theory of computation → Cell probe models and lower bounds

Keywords and phrases Gray code, Space-optimal counter, Decision assignment tree, Cell probe model

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.12

Related Version [8], `https://arxiv.org/abs/1712.01834`

Funding The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement no. 616787. The third author was also partially supported by the Center of Excellence CE-ITI under the grant P202/12/G061 of GA ČR.

© Diptarka Chakraborty, Debarati Das, Michal Koucký, and Nitin Saurabh;
licensed under Creative Commons License CC-BY

Acknowledgements Authors would like to thank Gerth Stølting Brodal for bringing the problem to our attention, and to Petr Gregor for giving a talk on space-optimal counters in our seminar, which motivated this research. Authors thank Meena Mahajan and Venkatesh Raman for pointing out the result in [39]. We also thank anonymous reviewers for helpful suggestions that improved the presentation of the paper.

1 Introduction

One of the fundamental problems in the domain of algorithm design is to list down all the objects belonging to a certain combinatorial class. Researchers are interested in efficient generation of a list such that an element in the list can be obtained by a small amount of change to the element that precedes it. One of the classic examples is the binary *Gray code* introduced by Gray [23], initially used in pulse code communication. The original idea of a Gray code was to list down all binary strings of length n, i.e, all the elements of $\mathbb{Z}_2^n$, such that any two successive strings differ by exactly one bit. The idea was later generalized for other combinatorial classes (e.g. see [37, 28]). Gray codes have found applications in a wide variety of areas, such as information storage and retrieval [9], processor allocation [10], computing the permanent [37], circuit testing [41], data compression [40], graphics and image processing [1], signal encoding [32], modulation schemes for flash memories [26, 22, 44] and many more. Interested reader may refer to an excellent survey by Savage [42] for a comprehensive treatment on this subject.

In this paper we study the construction of Gray codes over $\mathbb{Z}_m^n$ for any $m \in \mathbb{N}$. Originally, Gray codes were meant to list down all the elements from its domain but later studies (e.g. [20, 38, 5, 6]) focused on the generalization where we list ℓ distinct elements from the domain, each two consecutive elements differing in one position. We refer to such codes as Gray codes of length ℓ [20]. When the code lists all the elements from its domain it is referred to as *space-optimal*. It is often required that the last and the first strings appearing in the list also differ in one position. Such codes are called *cyclic Gray codes*. Throughout this paper we consider only cyclic Gray codes and we refer to them simply as Gray codes. Researchers also study codes where two successive strings differ in at most c positions, for some fixed constant $c > 0$, instead of differing in exactly one position. Such codes are called *quasi-Gray codes* [5][1] or c-Gray codes.

We study the problem of constructing quasi-Gray codes over $\mathbb{Z}_m^n$ in the cell probe model [43], where each cell stores an element from $\mathbb{Z}_m$. The efficiency of a construction is measured using three parameters. First, we want the length of a quasi-Gray code to be as large as possible. Ideally, we want space-optimal codes. Second, we want to minimize the number of coordinates of the input string the algorithm reads in order to generate the next (or, previous) string in the code. Finally, we also want the number of cells written in order to generate the successor (or, predecessor) string to be as small as possible. Since our focus is on quasi-Gray codes, the number of writes will always be bounded by a universal constant. We are interested in the worst-case behavior and we use *decision assignment trees* (DAT) of Fredman [20] to measure these complexities.

The second requirement of the above is motivated from the study of *loopless generation* of combinatorial objects. In the loopless generation we are required to generate the next string from the code in constant time. Different loopless algorithms to generate Gray codes

[1] Readers may note that the definition of quasi-Gray code given in [20] was different. The code referred as quasi-Gray code by Fredman [20] is called Gray code of length ℓ where $\ell < m^n$, in our notation.

are known in the literature [17, 4, 28]. However, those algorithms use extra memory cells in addition to the space required for the input string which makes it impossible to get a space-optimal code from them. More specifically, our goal is to design a decision assignment tree on n variables to generate a code over the domain $\mathbb{Z}_m^n$. If we allow extra memory cells (as in the case of loopless algorithms) then the corresponding DAT will be on $n + b$ variables, where b is the number of extra memory cells used.

Although there are known quasi-Gray codes with logarithmic read complexity and constant write complexity [20, 38, 5, 6], none of these constructions is space-optimal. The best result misses at least 2^{n-t} strings from the domain when having read complexity $t+O(\log n)$ [6]. Despite of an extensive research under many names, e.g., construction of Gray codes [20, 36, 16, 24], dynamic language membership problem [19], efficient representation of integers [38, 6], so far we do not have any quasi-Gray code of length $2^n - 2^{\epsilon n}$, for some constant $\epsilon < 1$, with worst-case read complexity $o(n)$ and write complexity $o(n)$. The best worst-case read complexity for *space-optimal* Gray code is $n - 1$ [21]. Recently, Raskin [39] showed that any space-optimal quasi-Gray code over the domain $\mathbb{Z}_2^n$ must have read complexity $\Omega(n)$. This lower bound is true even if we allow non-constant write complexity. It is worth noting that this result can be extended to the domain $\mathbb{Z}_m^n$ when m is even.

In this paper we show that such lower bound does not hold for quasi-Gray codes over $\mathbb{Z}_m^n$, when m is odd. In particular, we construct space-optimal quasi-Gray codes over $\{0, 1, 2\}^n$ with read complexity $4 \log_3 n$ and write complexity 2. As a consequence we get an exponential separation between the read complexity of space-optimal quasi-Gray code over $\mathbb{Z}_2^n$ and that over $\mathbb{Z}_3^n$.

▶ **Theorem 1.** *Let $m \in \mathbb{N}$ be odd and $n \in \mathbb{N}$ be such that $n \geq 15$. Then, there is a space-optimal quasi-Gray code C over $\mathbb{Z}_m^n$ for which, the two functions $next(C, w)$ and $prev(C, w)$ can be implemented by inspecting at most $4 \log_m n$ cells while writing only 2 cells.*

In the statement of the above theorem, $next(C, w)$ denotes the element appearing after w in the cyclic sequence of the code C, and analogously, $prev(C, w)$ denotes the preceding element. Using the argument as in [20, 36] it is easy to see a lower bound of $\Omega(\log_m n)$ on the read complexity when the domain is $\mathbb{Z}_m^n$. Hence our result is optimal up to some small constant factor.

Raskin shows $\Omega(n)$ lower bound on the read complexity of space-optimal binary quasi-Gray codes. The existence of binary quasi-Gray codes with sub-linear read complexity of length $2^n - 2^{\epsilon n}$, for some constant $\epsilon < 1$, was open. Using a different technique than that used in the proof of Theorem 1 we get a quasi-Gray code over the binary alphabet which enumerates all but $O(n)$ many strings. This result generalizes to the domain $\mathbb{Z}_q^n$ for any prime power q.

▶ **Theorem 2.** *Let $n \geq 15$ be any natural number. Then, there is a quasi-Gray code C of length at least $2^n - 20n$ over $\mathbb{Z}_2^n$, such that the two functions $next(C, w)$ and $prev(C, w)$ can be implemented by inspecting at most $6 + \log n$ cells while writing only 2 cells.*

We remark that the points that are missing from C in the above theorem are all of the form $\{0, 1\}^{O(\log n)} 0^{n-O(\log n)}$.

If we are allowed to read and write constant fraction of n bits then Theorem 2 can be adapted to get a quasi-Gray code of length $2^n - O(1)$ (see Section 5). In this way we get a trade-off between length of the quasi-Gray code and the number of bits read in the worst-case. All of our constructions can be made uniform.

Using the Chinese Remainder Theorem (cf. [14]), we also develop a technique that allows us to compose Gray codes over various domains. Hence, from quasi-Gray codes over domains

$\mathbb{Z}_{m_1}^n, \mathbb{Z}_{m_2}^n, \ldots, \mathbb{Z}_{m_k}^n$, where m_i's are pairwise co-prime, we can construct quasi-Gray codes over $\mathbb{Z}_m^{n'}$, where $m = m_1 \cdot m_2 \cdots m_k$. Using this technique on our main results, we get a quasi-Gray code over $\mathbb{Z}_m^n$ for *any* $m \in \mathbb{N}$ that misses only $O(n \cdot o^n)$ strings where $m = 2^\ell o$ for an odd o, while achieving the read complexity similar to that stated in Theorem 1. It is worth mentioning that if we get a space-optimal quasi-Gray code over the binary alphabet with non-trivial savings in read complexity, then we will have a space-optimal quasi-Gray code over the strings of alphabet $\mathbb{Z}_m$ for any $m \in \mathbb{N}$ with similar savings.

The technique by which we construct our quasi-Gray codes relies heavily on simple algebra which is a substantial departure from previous mostly combinatorial constructions. We view Gray codes as permutations on $\mathbb{Z}_m^n$ and we decompose them into k simpler permutations on $\mathbb{Z}_m^n$, each being computable with read complexity 3 and write complexity 1. Then we apply a different composition theorem, than mentioned above, to obtain space-optimal quasi-Gray codes on $\mathbb{Z}_m^{n'}$, $n' = n + \log k$, with read complexity $O(1) + \log k$ and write complexity 2. The main issue is the decomposition of permutations into few simple permutations. This is achieved by techniques of catalytic computation [7] going back to the work of Coppersmith and Grossman [13, 2, 3].

It follows from the work of Coppersmith and Grossman [13] that our technique is incapable of designing a space-optimal quasi-Gray code on $\mathbb{Z}_2^{n'}$ as any such code represents an *odd* permutation. The tools we use give inherently only *even* permutations. However, we can construct quasi-Gray codes from cycles of length $2^n - 1$ on $\mathbb{Z}_2^n$ as they are even permutations. Indeed, that is what we do for our Theorem 2. We note that any efficiently computable odd permutation on $\mathbb{Z}_2^n$, with say read complexity $(1-\epsilon)n$ and write complexity $O(1)$, could be used together with our technique to construct a space-optimal quasi-Gray code on $\mathbb{Z}_2^{n'}$ with read complexity at most $(1-\epsilon')n'$ and constant write complexity. This would represent a major progress on space-optimal Gray codes. (We would compose the odd permutation with some even permutation to obtain a full cycle on $\mathbb{Z}_2^n$. The size of the decomposition of the even permutation into simpler permutations would govern the read complexity of the resulting quasi-Gray code.)

Interestingly, Raskin's result relies on showing that a decision assignment tree of sub-linear read complexity must compute an even permutation.

1.1 Related works

The construction of Gray codes is central to the design of algorithms for many combinatorial problems [42]. Frank Gray [23] first came up with a construction of Gray code over binary strings of length n, where to generate the successor or predecessor strings one needs to read n bits in the worst-case. The type of code described in [23] is known as *binary reflected Gray code*. Later Bose *et al.* [5] provided a different type of Gray code construction, namely *recursive partition Gray code* which attains $O(\log n)$ average case read complexity while having the same worst-case read requirements. The read complexity we referred here is in the bit-probe model. It is easy to observe that any space-optimal binary Gray code must read $\log n + 1$ bits in the worst-case [20, 36, 21]. Recently, this lower bound was improved to $n/2$ in [39]. An upper bound of even $n-1$ was not known until very recently [21]. This is also the best known so far.

Fredman [20] extended the definition of Gray codes by considering codes that may not enumerate all the strings (though presented in a slightly different way in [20]) and also introduced the notion of *decision assignment tree* (DAT) to study the complexity of any code in the bit-probe model. He provided a construction that generates a Gray code of length $2^{c \cdot n}$ for some constant $c < 1$ while reducing the worst-case bit-read to $O(\log n)$. Using

Table 1 Taxonomy of construction of Gray/quasi-Gray codes over $\mathbb{Z}_m^n$.

Reference	Value of m	length	Worst-case cell read	Worst-case cell write
[23]	2	2^n	n	1
[20]	2	$2^{\Theta(n)}$	$O(\log n)$	$O(1)$
[19]	2	$\Theta(2^n/n)$	$\log n + 1$	$\log n + 1$
[38]	2	2^{n-1}	$\log n + 4$	4
[5]	2	$2^n - O(2^n/n^t)$	$O(t \log n)$	3
[6]	2	$2^n - 2^{n-t}$	$\log n + t + 3$	2
[6]	2	$2^n - 2^{n-t}$	$\log n + t + 2$	3
[21]	2	2^n	$n - 1$	1
Theorem 2	2	$2^n - O(n)$	$\log n + 4$	2
[12]	any m	m^n	n	1
Theorem 1	any odd m	m^n	$4 \log_m n + 3$	2

the idea of Lucal's modified reflected binary code [31], Munro and Rahman [38] got a code of length 2^{n-1} with worst-case read complexity only $4 + \log n$. However in their code two successive strings differ by 4 coordinates in the worst-case, instead of just one and we refer to such codes as quasi-Gray codes following the nomenclature used in [5]. Brodal et al. [6] extended the results of [38] by constructing a quasi-Gray code of length $2^n - 2^{n-t}$ for arbitrary $1 \leq t \leq n - \log n - 1$, that has $t + 3 + \log n$ bits ($t + 2 + \log n$ bits) worst-case read complexity and any two successive strings in the code differ by at most 2 bits (3 bits).

In contrast to the Gray codes over binary alphabets, Gray codes over non-binary alphabets received much less attention. The construction of binary reflected Gray code was generalized to the alphabet $\mathbb{Z}_m$ for any $m \in \mathbb{N}$ in [18, 12, 27, 40, 28, 25]. However, each of those constructions reads n coordinates in the worst-case to generate the next element. As mentioned before, we measure the read complexity in the well studied cell probe model [43] where we assume that each cell stores an element of $\mathbb{Z}_m$. The argument of Fredman in [20] implies a lower bound of $\Omega(\log_m n)$ on the read complexity of quasi-Gray code on $\mathbb{Z}_m^n$. To the best of our knowledge, for non-binary alphabets, there is nothing known similar to the results of Munro and Rahman or Brodal et al. [38, 6]. We summarize the previous results along with ours in Table 1.

Additionally, many variants of Gray codes have been studied in the literature. A particular one that has garnered a lot of attention in the past 30 years is the well-known *middle levels conjecture*. See [33, 34, 35, 24], and the references therein. It has been established only recently [33]. The conjecture says that there exists a Hamiltonian cycle in the graph induced by the vertices on levels n and $n + 1$ of the hypercube graph in $2n + 1$ dimensions. In other words, there exists a Gray code on the middle levels. Mütze et al. [34, 35] studied the question of efficiently enumerating such a Gray code in the word RAM model. They [35] gave an algorithm to enumerate a Gray code in the middle levels that requires $O(n)$ space and *on average* takes $O(1)$ time to generate the next vertex. In this paper we consider the bit-probe model, and Gray codes over the complete hypercube. It would be interesting to know whether our technique can be applied for the middle level Gray codes.

1.2 Our technique

Our construction of Gray codes relies heavily on the notion of s-functions defined by Coppersmith and Grossman [13]. An s-function is a permutation τ on $\mathbb{Z}_m^n$ defined by a function $f : \mathbb{Z}_m^s \to \mathbb{Z}_m$ and an $(s + 1)$-tuple of indices $i_1, i_2, \ldots, i_s, j \in [n]$ such that

$\tau(\langle x_1, x_2, \ldots, x_n\rangle) = (\langle x_1, x_2, \ldots, x_{j-1}, x_j + f(x_{i_1}, \ldots, x_{i_s}), x_{j+1}, \ldots, x_n\rangle)$, where the addition is inside $\mathbb{Z}_m$. Each s-function can be computed by some decision assignment tree that given a vector $x = \langle x_1, x_2, \ldots, x_n\rangle$, inspects $s+1$ coordinates of x and then it writes into a single coordinate of x.

A counter C (quasi-Gray code) on $\mathbb{Z}_m^n$ can be thought of as a permutation on $\mathbb{Z}_m^n$. Our goal is to construct some permutation α on $\mathbb{Z}_m^n$ that can be written as a composition of 2-functions $\alpha_1, \ldots, \alpha_k$, i.e., $\alpha = \alpha_k \circ \alpha_{k-1} \circ \cdots \circ \alpha_1$.

Given such a decomposition, we can build another counter C' on $\mathbb{Z}_m^{r+n}$, where $r = \lceil \log_m k \rceil$, for which the function $next(C', x)$ operates as follows. The first r-coordinates of x serve as an *instruction pointer* $i \in [m^r]$ that determines which α_i should be executed on the remaining n coordinates of x. Hence, based on the current value i of the r coordinates, we perform α_i on the remaining coordinates and then we update the value of i to $i+1$. (For $i > k$ we can execute the identity permutation which does nothing.)

We can use known Gray codes on $\mathbb{Z}_m^r$ to represent the instruction pointer so that when incrementing i we only need to write into one of the coordinates. This gives a counter C' which can be computed by a decision assignment tree that reads $r+3$ coordinates and writes into 2 coordinates of x. (A similar composition technique is implicit in Brodal et al. [6].) If C is of length $\ell = m^n - t$, then C' is of length $m^{n+r} - tm^r$. In particular, if C is space-optimal then so is C'.

Hence, we reduce the problem of constructing 2-Gray codes to the problem of designing large cycles in $\mathbb{Z}_m^n$ that can be decomposed into 2-functions. Coppersmith and Grossman [13] studied precisely the question of, which permutations on $\mathbb{Z}_2^n$ can be written as a composition of 2-functions. They show that a permutation on $\mathbb{Z}_2^n$ can be written as a composition of 2-functions if and only if the permutation is even. Since $\mathbb{Z}_2^n$ is of even size, a cycle of length 2^n on $\mathbb{Z}_2^n$ is an odd permutation and thus it cannot be represented as a composition of 2-functions. However, their result also implies that a cycle of length $2^n - 1$ on $\mathbb{Z}_2^n$ *can* be decomposed into 2-functions.

We want to use the counter composition technique described above in connection with a cycle of length $2^n - 1$. To maximize the length of the cycle C' in $\mathbb{Z}_2^{n+r}$, we need to minimize k, the number of 2-functions in the decomposition. By a simple counting argument, most cycles of length $2^n - 1$ on $\mathbb{Z}_2^n$ require k to be exponentially large in n. This is too large for our purposes. Luckily, there are cycles of length $2^n - 1$ on $\mathbb{Z}_2^n$ that can be decomposed into polynomially many 2-functions, and we obtain such cycles from linear transformations.

There are linear transformations $\mathbb{Z}_2^n \to \mathbb{Z}_2^n$ which define a cycle on $\mathbb{Z}_2^n$ of length $2^n - 1$. For example, the matrix corresponding to the multiplication by a fixed generator of the multiplicative group $\mathbb{F}_{2^n}^*$ of the Galois field $GF[2^n]$ is such a matrix. Such matrices are full rank and they can be decomposed into $O(n^2)$ elementary matrices, each corresponding to a 2-function. Moreover, there are matrices derived from primitive polynomials that can be decomposed into at most $4n$ elementary matrices.[2] We use them to get a counter on $\mathbb{Z}_2^{n'}$ of length at least $2^{n'} - 20n'$ whose successor and predecessor functions are computable by decision assignment trees of read complexity $\leq 6 + \log n'$ and write complexity 2. Such counter represents 2-Gray code of the prescribed length. For any prime q, the same construction yields 2-Gray codes of length at least $q^{n'} - 5q^2 n'$ with decision assignment trees of read complexity $\leq 6 + \log_q n'$ and write complexity 2.

[2] Primitive polynomials were previously also used in a similar problem, namely to construct shift-register sequences (see e.g. [28]).

The results of Coppersmith and Grossman [13] can be generalized to $\mathbb{Z}_m^n$ as stated in Richard Cleve's thesis [11].[3] For odd m, if a permutation on $\mathbb{Z}_m^n$ is even then it can be decomposed into 2-functions. Since m^n is odd, a cycle of length m^n on $\mathbb{Z}_m^n$ is an even permutation and so it can be decomposed into 2-functions. If the number k of those functions is small, so the $\log_m k$ is small, we get the sought after counter with small read complexity. However, for most cycles of length m^n on $\mathbb{Z}_m^n$, k is exponential in n.

We show though, that there is a cycle α of length m^n on $\mathbb{Z}_m^n$ that can be decomposed into $O(n^3)$ 2-functions. This in turn gives space-optimal 2-Gray codes on $\mathbb{Z}_m^{n'}$ with decision assignment trees of read complexity $O(\log_m n')$ and write complexity 2.

We obtain the cycle α and its decomposition in two steps. First, for $i \in [n]$, we consider the permutation α_i on $\mathbb{Z}_m^n$ which maps each element $0^{i-1}ay$ onto $0^{i-1}(a+1)y$, for $a \in \mathbb{Z}_m$ and $y \in \mathbb{Z}_m^{n-i}$, while other elements are mapped to themselves. Hence, α_i is a product of m^{n-i} disjoint cycles of length m. We show that $\alpha = \alpha_n \circ \alpha_{n-1} \circ \cdots \circ \alpha_1$ is a cycle of length m^n. In the next step we decompose each α_i into $O(n^2)$ 2-functions.

For $i \leq n-2$, we can decompose α_i using the technique of Ben-Or and Cleve [3] and its refinement in the form of catalytic computation of Buhrman et al. [7]. We can think of $x \in \mathbb{Z}_m^n$ as content of n memory registers, where $x_1, \ldots, x_{i-1}$ are the input registers, x_i is the output register, and $x_{i+1}, \ldots, x_n$ are the working registers. The catalytic computation technique gives a program consisting of $O(n^2)$ instructions, each being equivalent to a 2-function, which performs the desired adjustment of x_i based on the values of $x_1, \ldots, x_{i-1}$ without changing the ultimate values of the other registers. (We need to increment x_i iff $x_1, \ldots, x_{i-1}$ are all zero.) This program directly gives the desired decomposition of α_i, for $i \leq n-2$. (Our proof in Section 6 uses the language of permutations.)

The technique of catalytic computation fails for α_{n-1} and α_n as the program needs at least two working registers to operate. Hence, for α_{n-1} and α_n we have to develop entirely different technique. This is not trivial and quite technical but it is nevertheless possible, thanks to the specific structure of α_{n-1} and α_n.

2 Preliminaries

In the rest of the paper we only present constructions of the successor function $next(C, w)$ for our codes. Since all the operations in those constructions are readily invertible, the same arguments also give the predecessor function $prev(C, w)$.

Notation: We use the standard notions of groups and fields, and mostly we will use only elementary facts about them (see [15, 30] for background.). By $\mathbb{Z}_m$ we mean the set of integers modulo m, i.e., $\mathbb{Z}_m := \mathbb{Z}/m\mathbb{Z}$. Throughout this paper whenever we use addition and multiplication operation between two elements of $\mathbb{Z}_m$, then we mean the operations within $\mathbb{Z}_m$ that is modulo m. For any $m \in \mathbb{N}$, we let $[m]$ denote the set $\{1, 2, \ldots, m\}$. Unless stated otherwise explicitly, all the logarithms we consider throughout this paper are based 2.

Now we define the notion of counters used in this paper.

▶ **Definition 3** (Counter)**.** A *counter* of length ℓ over a domain $\mathcal{D}$ is any cyclic sequence $C = (w_1, \ldots, w_\ell)$ such that $w_1, \ldots, w_\ell$ are distinct elements of $\mathcal{D}$. With the counter C we associate two functions $next(C, w)$ and $prev(C, w)$ that give the successor and predecessor element of w in C, that is for $i \in [\ell]$, $next(C, w_i) = w_j$ where $j - i = 1 \bmod \ell$, and $prev(C, w_i) = w_k$ where $i - k = 1 \bmod \ell$. If $\ell = |\mathcal{D}|$, we call the counter a *space-optimal counter*.

[3] Unfortunately, there is no written record of the proof.

Often elements in the underlying domain $\mathcal{D}$ have some "structure" to them. In such cases, it is desirable to have a counter such that consecutive elements in the sequence differ by a "small" change in the "structure". We make this concrete in the following definition.

▶ **Definition 4** (Gray Code). Let $\mathcal{D}_1, \dots, \mathcal{D}_n$ be finite sets. A *Gray code* of length ℓ over the domain $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_n$ is a counter $(w_1, \dots, w_\ell)$ of length ℓ over $\mathcal{D}$ such that any two consecutive strings w_i and w_j, $j - i = 1 \bmod \ell$, differ in exactly one coordinate when viewed as an n-tuple. More generally, if for some constant $c \geq 1$, any two consecutive strings w_i and w_j, $j - i = 1 \bmod \ell$, differ in at most c coordinates such a counter is called a *c-Gray Code.*

By a quasi-Gray code we mean c-Gray code for some unspecified fixed $c > 0$. In the literature sometimes people do not place any restriction on the relationship between w_ℓ and w_1 and they refer to such a sequence a (quasi)-Gray code. In their terms, our codes would be *cyclic* (quasi)-Gray codes. If $\ell = |\mathcal{D}|$, we call the codes *space-optimal* (quasi-)Gray codes.

Decision Assignment Tree: The computational model we consider in this paper is called *Decision Assignment Tree* (DAT). The definition we provide below is a generalization of that given in [20]. It is intended to capture random access machines with small word size.

Let us fix an underlying domain $\mathcal{D}^n$ whose elements we wish to enumerate. In the following, we will denote an element in $\mathcal{D}^n$ by $\langle x_1, x_2, \dots, x_n\rangle$. A decision assignment tree is a $|\mathcal{D}|$-ary tree such that each internal node is labeled by one of the variables $x_1, x_2, \dots, x_n$. Furthermore, each outgoing edge of an internal node is labeled with a distinct element of $\mathcal{D}$. Each leaf node of the tree is labeled by a set of assignment instructions that set new (fixed) values to chosen variables. The variables which are not mentioned in the assignment instructions remain unchanged.

The execution on a decision assignment tree on a particular input vector $\langle x_1, \dots, x_n\rangle \in \mathcal{D}^n$ starts from the root of the tree and continues in the following way: at a non-leaf node labeled with a variable x_i, the execution queries x_i and depending on the value of x_i the control passes to the node following the outgoing edge labeled with the value of x_i. Upon reaching a leaf, the corresponding set of assignment statements is used to modify the vector $\langle x_1, x_2, \dots, x_n\rangle$ and the execution terminates. The modified vector is the output of the execution.

Thus, each decision assignment tree computes a mapping from $\mathcal{D}^n$ into $\mathcal{D}^n$. We are interested in decision assignment trees computing the mapping $next(C, \langle x_1, x_2, \dots, x_n\rangle)$ for some counter C. When C is space-optimal we can assume, without loss of generality, that each leaf assigns values only to the variables that it reads on the path from the root to the leaf. (Otherwise, the decision assignment tree does not compute a bijection.) We define the *read complexity* of a decision assignment tree T, denoted by $\mathrm{READ}(T)$, as the maximum number of non-leaf nodes along any path from the root to a leaf. Observe that any mapping from $\mathcal{D}^n$ into $\mathcal{D}^n$ can be implemented by a decision assignment tree with read complexity n. We also define the *write complexity* of a decision assignment tree T, denoted by $\mathrm{WRITE}(T)$, as the maximum number of assignment instructions in any leaf.

Instead of the domain $\mathcal{D}^n$, we will sometimes also use domains that are a cartesian product of different domains. The definition of a decision assignment tree naturally extends to this case of different variables having different domains.

For any counter $C = (w_1, \dots, w_\ell)$, we say that C is computed by a decision assignment tree T if and only if for $i \in [\ell]$, $next(C, w_i) = T(w_i)$, where $T(w_i)$ denotes the output string obtained after an execution of T on w_i. Note that any two consecutive strings in the cyclic sequence of C differ by at most $\mathrm{WRITE}(T)$ many coordinates. For a small constant $c \geq 1$, some domain $\mathcal{D}$, and all large enough n, we will be interested in constructing cyclic counters on $\mathcal{D}^n$ that are computed by decision assignment trees of write complexity c and read complexity $O(\log n)$. By definition such cyclic counters will necessarily be c-Gray codes.

2.1 Construction of Gray codes

For our construction of quasi-Gray codes on a domain $\mathcal{D}^n$ with decision assignment trees of small read and write complexity we will need ordinary Gray codes on a domain $\mathcal{D}^{O(\log n)}$. Several constructions of space-optimal binary Gray codes are known where the oldest one is the binary reflected Gray code [23]. This can be generalized to space-optimal (cyclic) Gray codes over non-binary alphabets (see e.g. [12, 28]).

▶ **Theorem 5** ([12, 28])**.** *For any $m, n \in \mathbb{N}$, there is a space-optimal (cyclic) Gray code over $\mathbb{Z}_m^n$.*

3 Chinese Remainder Theorem for Counters

Below we describe how to compose decision assignment trees over different domains to get a decision assignment tree for a larger mixed domain. For all the details and proofs we refer the reader to the full version of this paper [8].

▶ **Theorem 6** (Chinese Remainder Theorem for Counters)**.** *Let $r, n_1, \ldots, n_r \in \mathbb{N}$ be integers, and let $\mathcal{D}_{1,1}, \ldots, \mathcal{D}_{1,n_1}, \mathcal{D}_{2,1}, \ldots, \mathcal{D}_{r,n_r}$ be some finite sets of size at least two. Let $\ell_1 \geq r-1$ be an integer, and $\ell_2, \ldots, \ell_r$ be pairwise co-prime integers. For $i \in [r]$, let C_i be a counter of length ℓ_i over $\mathcal{D}_i = \mathcal{D}_{i,1} \times \cdots \times \mathcal{D}_{i,n_i}$ computed by a decision assignment tree T_i over n_i variables. Then, there exists a decision assignment tree T over $\sum_{i=1}^r n_i$ variables that implements a counter C of length $\prod_{i=1}^r \ell_i$ over $\mathcal{D}_1 \times \cdots \times \mathcal{D}_r$. Furthermore, $\mathrm{READ}(T) = n_1 + \max\{\mathrm{READ}(T_i)\}_{i=2}^r$, and $\mathrm{WRITE}(T) = \mathrm{WRITE}(T_1) + \max\{\mathrm{WRITE}(T_i)\}_{i=2}^r$.*

We remark that if C_i's in the theorem are space-optimal then so is C. The proof of the theorem constructs a special type of a counter where we always read the first coordinate, increment it, and further depending on its value, we may update the value of another coordinate. Note, for such type of a counter the co-primality condition is necessary at least for $\ell_1 = 2, 3$ (see the full version [8]).

As a corollary of the above theorem, to get a decision assignment tree implementing space-optimal quasi-Gray codes over $\mathbb{Z}_m$ for any $m \in \mathbb{N}$, we only need decision assignment trees implementing space-optimal quasi-Gray codes over $\mathbb{Z}_2$ and $\mathbb{Z}_m$, for odd m.

4 Permutation Group and Construction of Counters

We start this section with some basic notation and facts about the permutation group which we will use heavily in the rest of the paper. The set of all permutations over a domain $\mathcal{D}$ forms a group under the composition operation, denoted by $\circ$, which is defined as follows: for any two permutations σ and α, $\sigma \circ \alpha(x) = \sigma(\alpha(x))$, where $x \in \mathcal{D}$. The corresponding group, denoted $\mathcal{S}_N$, is the *symmetric group* of order $N = |\mathcal{D}|$. We say, a permutation $\sigma \in \mathcal{S}_N$ is a *cycle* of length ℓ if there are distinct elements $a_1, \ldots, a_\ell \in [N]$ such that for $i \in [\ell - 1]$, $a_{i+1} = \sigma(a_i)$, $a_1 = \sigma(a_\ell)$, and for all $a \in [N] \setminus \{a_1, a_2, \ldots, a_\ell\}$, $\sigma(a) = a$. We denote such a cycle by $(a_1, a_2, \cdots, a_\ell)$.

Roughly speaking, a counter of length ℓ over $\mathcal{D}$, in the language of permutations, is nothing but a cycle of the same length in $\mathcal{S}_{|\mathcal{D}|}$. We now make this correspondence precise and give a construction of a decision assignment tree that implements such a counter.

We state our key lemma to construct Gray codes from a decomposition of a permutation.

▶ **Lemma 7.** *Let $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_r$ be a domain. Suppose $\sigma_1, \ldots, \sigma_k \in \mathcal{S}_{|\mathcal{D}|}$ are such that $\sigma = \sigma_k \circ \sigma_{k-1} \circ \cdots \circ \sigma_1$ is a cycle of length ℓ. Let $T_1, \ldots, T_k$ be decision assignment trees that implement $\sigma_1, \ldots, \sigma_k$ respectively. Let $\mathcal{D}' = \mathcal{D'}_1 \times \cdots \times \mathcal{D'}_{r'}$ be a domain such that $|\mathcal{D}'| \geq k$,* *and let T' be a decision assignment tree that implements a counter C' of length k' over $\mathcal{D}'$ where $k' \geq k$.*

Then, there exists a decision assignment tree T that implements a counter of length $k'\ell$ over $\mathcal{D}' \times \mathcal{D}$ such that $\mathrm{READ}(T) = r' + \max\{\mathrm{READ}(T_i)\}_{i=1}^{k}$, *and* $\mathrm{WRITE}(T) = \mathrm{WRITE}(T') + \max\{\mathrm{WRITE}(T_i)\}_{i=1}^{k}$.

Proof sketch. Suppose $C' = (a_1, \ldots, a_{k'})$. Now let us consider the following procedure P: on any input $\langle x_1, x_2 \rangle \in \mathcal{D}' \times \mathcal{D}$, if $x_1 = a_j$ for some $j \in [k]$, set $x_2 \leftarrow \sigma_j(x_2)$. Next, increment the first coordinate, i.e., set $x_1 \leftarrow \mathit{next}(C', x_1)$.

Read and write complexity of the statement follows immediately. The correctness also follows from some basic property of permutation group together with the fact that σ is a cycle of length ℓ. ◀

In the next two sections we describe the construction of $\sigma_1, \cdots, \sigma_k \in \mathcal{S}_N$ where $N = m^n$ for some $m, n \in \mathbb{N}$ and how the value of k depends on the length of the cycle $\sigma = \sigma_k \circ \sigma_{k-1} \circ \cdots \circ \sigma_1$.

5 Counters via Linear Transformation

The construction in this section is based on linear transformations. Consider the vector space $\mathbb{F}_q^n$, and let $L : \mathbb{F}_q^n \to \mathbb{F}_q^n$ be a linear transformation. A basic fact in linear algebra says that if L has *full* rank, then the mapping given by L is a bijection. Thus, when L is full rank, the mapping can also be thought of as a permutation over $\mathbb{F}_q^n$. Throughout this section we use many basic terms related to linear transformation without defining them, for the details of which we refer the reader to any standard text book on linear algebra (e.g. [29]).

A natural way to build counter out of a full rank linear transformation is to fix a starting element, and repeatedly apply the linear transformation to obtain the next element (cf. [28]). Clearly this only list out elements in the cycle containing the starting element. Therefore, we would like to choose the starting element such that we enumerate the largest cycle. Ideally, we would like the largest cycle to contain all the elements of $\mathbb{F}_q^n$. However this is not possible because any linear transformation fixes the all-zero vector. But there do exist full rank linear transformations such that the permutation given by them is a single cycle of length $q^n - 1$. Such a linear transformation would give us a counter over a domain of size q^n that enumerates all but one element. Clearly, a trivial implementation of the aforementioned argument would lead to a counter that reads and writes all n coordinates in the worst-case. In the rest of this section, we will develop an implementation and argue about the choice of linear transformation such that the read and write complexity decreases exponentially.

It is well known that every linear transformation L is associated with some matrix $A \in \mathbb{F}_q^{n \times n}$ such that applying the linear transformation is equivalent to the left multiplication by A. Furthermore, L has full rank iff A is invertible over $\mathbb{F}_q$.

▶ **Definition 8** (Elementary matrices)**.** An $n \times n$ matrix over a field $\mathbb{F}$ is said to be an *elementary matrix* if it has one of the following forms:

(a) The off-diagonal entries are all 0. For some $i \in [n]$, (i,i)-th entry is a non-zero $c \in \mathbb{F}$. Rest of the diagonal entries are 1.

(b) The diagonal entries are all 1. For some i and j, $1 \leq i \neq j \leq n$, (i,j)-th entry is a non-zero $c \in \mathbb{F}$. Rest of the off-diagonal entries are 0.

From the definition it is easy to see that left multiplication by an elementary matrix of the first type is equivalent to multiplying the i-th row with c, and by an elementary matrix of the second type it is equivalent to adding c times j-th row to the i-th row.

▶ **Proposition 9.** *Let $A \in \mathbb{F}^{n\times n}$ be invertible. Then A can be written as a product of k elementary matrices such that $k \leq n^2 + 4(n-1)$.*

The proof follows from Gaussian elimination.

5.1 Construction of the counter

Let A be a full rank linear transformation from $\mathbb{F}_q^n$ to $\mathbb{F}_q^n$ such that when viewed as permutation it is a single cycle of length $q^n - 1$. More specifically, A is an invertible matrix in $\mathbb{F}_q^{n\times n}$ such that for any $x \in \mathbb{F}_q^n$ where $x \neq (0,\ldots,0)$, $Ax, A^2x, \ldots, A^{(q^n-1)}x$ are distinct. Such a matrix exists, for example, take A to be the matrix of a linear transformation that corresponds to multiplication from left by a fixed generator of the multiplicative group of $\mathbb{F}_{q^n}$ under the standard vector representation of elements of $\mathbb{F}_{q^n}$. Let $A = E_k E_{k-1} \cdots E_1$ where E_i's are elementary matrices.

▶ **Theorem 10.** *Let q, A, and k be as defined above. Let $r \geq \log_q k$. There exists a quasi-Gray code on the domain $(\mathbb{F}_q)^{n+r}$ of length $q^{n+r} - q^r$ that can be implemented using a decision assignment tree T such that* $\mathrm{READ}(T) \leq r+2$ *and* $\mathrm{WRITE}(T) \leq 2$.

Proof. The proof follows readily from Lemma 7, where E_i's play the role of σ_i's, and noting that the permutation given by any elementary matrix can be implemented using a decision assignment tree that reads at most two coordinates and writes at most one. For the counter C' on $(\mathbb{F}_q)^r$ we chose a Gray code of trivial read complexity r and write complexity 1. ◀

Thus, we obtain a counter on a domain of size roughly kq^n that misses at most qk elements. Clearly, we would like to minimize k. A trivial bound on k is $O(n^2)$ that follows from Proposition 9. We now discuss the choice of A so that k becomes $O(n)$ based on primitive polynomials over finite fields.

Let $p(z)$ be a primitive polynomial of degree n over $\mathbb{F}_q$, where $p(z) = z^n + c_{n-1}z^{n-1} + c_{n-2}z^{n-2} + \cdots + c_1 z + c_0$. The matrix A defined as follows is the matrix representing multiplication by some generator (a root of $p(z)$) of the multiplicative group of $\mathbb{F}_{q^n}$:

$$\begin{pmatrix} -c_{n-1} & 1 & 0 & \cdots & 0 \\ -c_{n-2} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -c_1 & 0 & 0 & \cdots & 1 \\ -c_0 & 0 & 0 & \cdots & 0 \end{pmatrix}.$$

It is easy to see that A can be written as a product of at most $n + 4(n-1)$ elementary matrices. (In case, q is a power of 2, then the number of elementary matrices in the product is at most $n + 3(n-1)$.) Hence, from the discussion above and using Theorem 10, we obtain the following corollaries. Setting $r = \lceil \log(4n-3) \rceil$ in Theorem 10 gives:

▶ **Corollary 11.** *For any $n' \geq 2$, and $n = n' + \lceil \log(4n'-3) \rceil$, there exists a counter on $(\mathbb{Z}_2)^n$ that misses at most $8n$ strings and can be implemented by a decision assignment tree that reads at most $4 + \log n$ bits and writes at most 2 bits.*

By doubling the number of missed strings and increasing the number of read bits by one we can construct given counters for any $\mathbb{Z}_2^n$, where $n \geq 15$. For the general case, when q is a prime power, we obtain the following corollary by setting r to $\lceil \log_q(5n-4) \rceil$ or $1 + \lceil \log_q(5n-4) \rceil$ in Theorem 10.

▶ **Corollary 12** (Generalization of Theorem 2). *Let q be any prime power. For $n \geq 15$, there exists a counter on $\mathbb{Z}_q^n$ that misses at most $5q^2n$ strings and that is computed by a decision assignment tree with read complexity at most $6 + \log_q n$ and write complexity 2.*

6 Space-optimal Counters over $\mathbb{Z}_m^n$ for any Odd m

In this section we sketch a proof of Theorem 1. We want to use Lemma 7. Set $n' = n - c \cdot \log n$ for a suitable constant $c > 0$. We define permutations $\alpha_1, \cdots, \alpha_{n'} \in \mathcal{S}_N$, for $N = m^{n'}$, such that $\alpha = \alpha_{n'} \circ \cdots \circ \alpha_1$ is a cycle of length $m^{n'}$. We will show that each of these α_i's can be further decomposed into $\alpha_{i,1}, \cdots, \alpha_{i,j} \in \mathcal{S}_N$ for some j, such that each of $\alpha_{i,r}$ for $r \in [j]$ can be implemented using DAT with read complexity 3 and write complexity 1. Finally we use Lemma 7 by considering all these $\alpha_{i,r}$'s as $\sigma_1, \cdots, \sigma_k$, where k is $O(mn'^3)$.

For any $i \in [n']$, define α_i as follows: for any $\langle x_1, \cdots, x_{n'} \rangle \in \mathbb{Z}_m^{n'}$, if $x_j = 0$ for all $j = 1, \cdots, i-1$, then $x_i \leftarrow x_i + 1 \mod m$. Observe, $\alpha = \alpha_{n'} \circ \cdots \circ \alpha_1$ is a cycle of length $m^{n'}$. Notice each α_i is a $(i-1)$-function on $\mathbb{Z}_m^{n'}$ (see Section 1.2 for their definition). Now if for any $i \in [n']$ we can find a set of 2-functions $\alpha_{i,1}, \cdots, \alpha_{i,k_i}$ such that $\alpha_i = \alpha_{i,k_i} \circ \cdots \circ \alpha_{i,1}$, then we can use them as σ_j's in Lemma 7. As a result the complexity in Theorem 1 follows.

Note α_1, α_2 and α_3 are already 2-functions. In the case of α_i for $4 \leq i \leq n' - 2$, we can directly adopt the technique from [3, 7] to generate the desired set of 2-functions. For $i = n' - 1$, it is possible to generalize the proof technique of [13] to decompose $\alpha_{n'-1}$ but we have to develop a new technique to decompose $\alpha_{n'}$.

▶ **Lemma 13.** *For any $4 \leq i \leq n' - 2$, one can construct a set of 2-functions $\alpha_{i,1}, \cdots, \alpha_{i,k_i}$ such that $\alpha_i = \alpha_{i,k_i} \circ \cdots \circ \alpha_{i,1}$ where $k_i \leq c_1 \cdot (i-1)^2$ for some constant $c_1 > 0$.*

It remains to decompose $\alpha_{n'-1}$ and $\alpha_{n'}$. A key tool is the following lemma.

▶ **Lemma 14.** *Suppose there are two cycles, $\sigma = (t, a_1, \cdots, a_{\ell-1})$ and $\tau = (t, b_1, \cdots, b_{\ell-1})$, of length $\ell \geq 2$ such that $a_i \neq b_j$ for every $i, j \in [\ell - 1]$. Then, $(\sigma \circ \tau)^\ell \circ (\tau \circ \sigma)^\ell = \sigma^2$.*

Let us now consider $\alpha_{n'}$, the case of $\alpha_{n'-1}$ is analogous. For $a = (m+1)/2$, we define $\sigma = (\langle 00 \cdots 0(0 \cdot a) \rangle, \langle 00 \cdots 0(1 \cdot a) \rangle, \cdots, \langle 00 \cdots 0((m-1) \cdot a) \rangle)$, and $\tau = (\langle (0 \cdot a)00 \cdots 0 \rangle, \langle (1 \cdot a)00 \cdots 0 \rangle, \cdots, \langle ((m-1) \cdot a)00 \cdots 0 \rangle)$, where the multiplication is in $\mathbb{Z}_m$. In other words, we define σ by adding a to the value of the last coordinate when all other coordinates are set to 0, and we define τ by adding a to the value of the first coordinate when all other coordinates are set to 0. Since m is co-prime with $(m+1)/2$, σ and τ are cycles of length m. (Here we use the fact that m is odd.) Observe that $\sigma^2 = \alpha_{n'}$, so by applying Lemma 14 to σ and τ we get $\alpha_{n'}$. It might seem we didn't make much progress towards decomposition, as now instead of one $(n'-1)$-function $\alpha_{n'}$ we have to decompose two $(n'-1)$-functions σ and τ. However, we will not decompose σ and τ directly, but rather we obtain a decomposition for $(\sigma \circ \tau)^m$ and $(\tau \circ \sigma)^m$. Surprisingly this can be done using a generalization of Lemma 13.

We consider an $(n'-3)$-function σ' whose cycle decomposition contains σ as one of its cycles. Similarly we consider a 3-function τ' whose cycle decomposition contains τ as one of its cycles. We carefully choose these σ' and τ' such that $(\sigma' \circ \tau')^m = (\sigma \circ \tau)^m$ and $(\tau' \circ \sigma')^m = (\tau \circ \sigma)^m$. We will decompose σ' and τ' to get the desired decomposition.

▶ **Lemma 15.** *For any $i \in \{n'-1, n'\}$, one can construct a set of 2-functions $\alpha_{i,1}, \cdots, \alpha_{i,k_i}$ such that $\alpha_i = \alpha_{i,k_i} \circ \cdots \circ \alpha_{i,1}$ where $k_i \leq c_2 \cdot m \cdot (i-1)^2$ for some constant $c_2 > 0$.*

References

1 D. J. Amalraj, N. Sundararajan, and Goutam Dhar. Data structure based on Gray code encoding for graphics and image processing. In *Proceedings of the SPIE: International Society for Optical Engineering*, pages 65–76, 1990.

2 David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

3 Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.*, 21(1):54–58, 1992. doi:10.1137/0221006.

4 James R. Bitner, Gideon Ehrlich, and Edward M. Reingold. Efficient generation of the binary reflected Gray code and its applications. *Commun. ACM*, 19(9):517–521, 1976.

5 Prosenjit Bose, Paz Carmi, Dana Jansens, Anil Maheshwari, Pat Morin, and Michiel H. M. Smid. Improved methods for generating quasi-Gray codes. In *Algorithm Theory - SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory, Bergen, Norway, June 21-23, 2010. Proceedings*, pages 224–235, 2010. doi:10.1007/978-3-642-13731-0_22.

6 Gerth Stølting Brodal, Mark Greve, Vineet Pandey, and Srinivasa Rao Satti. Integer representations towards efficient counting in the bit probe model. *J. Discrete Algorithms*, 26:34–44, 2014. doi:10.1016/j.jda.2013.11.001.

7 Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 857–866, 2014. doi:10.1145/2591796.2591874.

8 Diptarka Chakraborty, Debarati Das, Michal Koucký, and Nitin Saurabh. Optimal quasi-Gray codes: Does the alphabet matter? *CoRR*, 2017. arXiv:1712.01834.

9 C. C. Chang, H. Y. Chen, and C. Y. Chen. Symbolic Gray code as a data allocation scheme for two-disc systems. *The Computer Journal*, 35(3):299–305, 1992. doi:10.1093/comjnl/35.3.299.

10 M. S. Chen and K. G. Shin. Subcube allocation and task migration in hypercube multiprocessors. *IEEE Transactions on Computers*, 39(9):1146–1155, 1990. doi:10.1109/12.57056.

11 Richard Cleve. *Methodologies for Designing Block Ciphers and Cryptographic Protocols*. PhD thesis, University of Toronto, April 1989.

12 Martin Cohn. Affine m-ary Gray codes. *Information and Control*, 6(1):70–78, 1963.

13 Don Coppersmith and Edna Grossman. Generators for certain alternating groups with applications to cryptography. *SIAM J. Appl. Math.*, 29(4):624–627, 1975.

14 C. Ding, D. Pei, and A. Salomaa. *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1996.

15 David S. Dummit and Richard M. Foote. *Abstract Algebra*. John Wiley & Sons, 2004.

16 Tomáš Dvořák, Petr Gregor, and Václav Koubek. Generalized Gray codes with prescribed ends. *Theor. Comput. Sci.*, 668:70–94, 2017. doi:10.1016/j.tcs.2017.01.010.

17 Gideon Ehrlich. Loopless algorithms for generating permutations, combinations, and other combinatorial configurations. *J. ACM*, 20(3):500–513, 1973. doi:10.1145/321765.321781.

18 Ivan Flores. Reflected number systems. *IRE Transactions on Electronic Computers*, EC-5(2):79–82, 1956.

19 Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *J. ACM*, 44(2):257–271, 1997. doi:10.1145/256303.256309.

20 Michael L. Fredman. Observations on the complexity of generating quasi-Gray codes. *SIAM J. Comput.*, 7(2):134–146, 1978. doi:10.1137/0207012.

21 Zachary Frenette. Towards the efficient generation of Gray codes in the bitprobe model. Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, 2016.

22 E. Gad, M. Langberg, M. Schwartz, and J. Bruck. Constant-weight Gray codes for local rank modulation. *IEEE Transactions on Information Theory*, 57(11):7431–7442, 2011. doi:10.1109/TIT.2011.2162570.

23 F. Gray. Pulse code communication, 1953. US Patent 2,632,058. URL: http://www.google.com/patents/US2632058.

24 Petr Gregor and Torsten Mütze. Trimming and gluing Gray codes. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 40:1–40:14, 2017. doi:10.4230/LIPIcs.STACS.2017.40.

25 Felix Herter and Günter Rote. Loopless Gray code enumeration and the tower of bucharest. In *8th International Conference on Fun with Algorithms, FUN 2016, June 8-10, 2016, La Maddalena, Italy*, pages 19:1–19:19, 2016.

26 A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck. Rank modulation for flash memories. *IEEE Transactions on Information Theory*, 55(6):2659–2673, 2009. doi:10.1109/TIT.2009.2018336.

27 James T. Joichi, Dennis E. White, and S. G. Williamson. Combinatorial Gray codes. *SIAM J. Comput.*, 9(1):130–141, 1980. doi:10.1137/0209013.

28 Donald E. Knuth. *The Art of Computer Programming. Volume 4A: Combinatorial Algorithms, Part 1.* Addison-Wesley Professional, 2011.

29 Serge Lang. *Linear Algebra.* Undergraduate Texts in Mathematics. Springer New York, 1987.

30 Rudolf Lidl and Harald Niederreiter. *Finite Fields.* Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2 edition, 1996. doi:10.1017/CBO9780511525926.

31 H. M. Lucal. Arithmetic operations for digital computers using a modified reflected binary code. *IRE Transactions on Electronic Computers*, EC-8(4):449–458, 1959.

32 J. Ludman. Gray code generation for mpsk signals. *IEEE Transactions on Communications*, 29(10):1519–1522, 1981. doi:10.1109/TCOM.1981.1094886.

33 Torsten Mütze. Proof of the middle levels conjecture. *Proceedings of the London Mathematical Society*, 112(4):677–713, 2016.

34 Torsten Mütze and Jerri Nummenpalo. Efficient computation of middle levels Gray codes. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 915–927, 2015.

35 Torsten Mütze and Jerri Nummenpalo. A constant-time algorithm for middle levels Gray codes. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2238–2253, 2017.

36 Patrick K. Nicholson, Venkatesh Raman, and S. Srinivasa Rao. A survey of data structures in the bitprobe model. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 303–318, 2013. doi:10.1007/978-3-642-40273-9_19.

37 Albert Nijenhuis and Herbert S. Wilf. *Combinatorial Algorithms.* Academic Press, 1978.

38 M. Ziaur Rahman and J. Ian Munro. Integer representation and counting in the bit probe model. *Algorithmica*, 56(1):105–127, 2010. doi:10.1007/s00453-008-9247-2.

39 Mikhail Raskin. A linear lower bound for incrementing a space-optimal integer representation in the bit-probe model. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 88:1–88:12, 2017.

40 Dana Richards. Data compression and Gray-code sorting. *Information Processing Letters*, 22(4):201–205, 1986. doi:10.1016/0020-0190(86)90029-3.

41 J. P. Robinson and M. Cohn. Counting sequences. *IEEE Transactions on Computers*, C-30(1):17–23, 1981. `doi:10.1109/TC.1981.6312153`.

42 Carla Savage. A survey of combinatorial Gray codes. *SIAM review*, 39(4):605–629, 1997.

43 Andrew Chi-Chih Yao. Should tables be sorted? *J. ACM*, 28(3):615–628, 1981. `doi:10.1145/322261.322274`.

44 Y. Yehezkeally and M. Schwartz. Snake-in-the-box codes for rank modulation. *IEEE Transactions on Information Theory*, 58(8):5471–5483, 2012. `doi:10.1109/TIT.2012.2196755`.

A Framework for In-place Graph Algorithms

Sankardeep Chakraborty[1]
The University of Tokyo, Japan
sankardeep@me2.mist.i.u-tokyo.ac.jp

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

Venkatesh Raman
The Institute of Mathematical Sciences, HBNI, India
vraman@imsc.res.in

Srinivasa Rao Satti
Seoul National University, South Korea
ssrao@cse.snu.ac.kr

Abstract

Read-only memory (ROM) model is a classical model of computation to study time-space tradeoffs of algorithms. A classical result on the ROM model is that any algorithm to sort n numbers using $O(s)$ words of extra space requires $\Omega(n^2/s)$ comparisons for $\lg n \leq s \leq n/\lg n^2$ and the bound has also been recently matched by an algorithm. However, if we relax the model, we do have sorting algorithms (say Heapsort) that can sort using $O(n \lg n)$ comparisons using $O(\lg n)$ bits of extra space, even keeping a permutation of the given input sequence at anytime during the algorithm.

We address similar relaxations for graph algorithms. We show that a simple natural relaxation of ROM model allows us to implement fundamental graph search methods like BFS and DFS more space efficiently than in ROM. By simply allowing elements in the adjacency list of a vertex to be permuted, we show that, on an undirected or directed connected graph G having n vertices and m edges, the vertices of G can be output in a DFS or BFS order using $O(\lg n)$ bits of extra space and $O(n^3 \lg n)$ time. Thus we obtain similar bounds for *reachability* and *shortest path distance* (both for undirected and directed graphs). With a little more (but still polynomial) time, we can also output vertices in the *lex-DFS* order. As reachability in directed graphs (even in DAGs) and shortest path distance (even in undirected graphs) are NL-complete, and lex-DFS is P-complete, our results show that our model is more powerful than ROM if $\mathsf{L} \neq \mathsf{P}$.

En route, we also introduce and develop algorithms for another relaxation of ROM where the adjacency lists of the vertices are circular lists and we can modify only the heads of the lists. Here we first show a linear time DFS implementation using $n + O(\lg n)$ bits of extra space. Improving the extra space exponentially to only $O(\lg n)$ bits, we also obtain BFS and DFS albeit with a slightly slower running time. Both the models we propose maintain the graph structure throughout the algorithm, only the order of vertices in the adjacency list changes. In sharp contrast, for BFS and DFS, to the best of our knowledge, there are no algorithms in ROM that use even $O(n^{1-\epsilon})$ bits of extra space; in fact, implementing DFS using cn bits for $c < 1$ has been mentioned as an open problem. Furthermore, DFS (BFS, respectively) algorithms using $n + o(n)$ ($o(n)$, respectively) bits of extra use Reingold's [JACM, 2008] or Barnes et al's reachability algorithm [SICOMP, 1998] and hence have high runtime. Our results can be contrasted with the recent result of Buhrman et al. [STOC, 2014] which gives an algorithm for *directed st-reachability* on *catalytic Turing machines* using $O(\lg n)$ bits with catalytic space $O(n^2 \lg n)$ and time $O(n^9)$.

[1] This work was partially supported by JST CREST Grant Number JPMJCR1402, Japan.
[2] We use lg to denote logarithm to the base 2.

© Sankardeep Chakraborty, Anish Mukherjee, Venkatesh Raman, and Srinivasa Rao Satti;
licensed under Creative Commons License CC-BY
26th Annual European Symposium on Algorithms (ESA 2018).
Editors: Yossi Azar, Hannah Bast, and Grzegorz Herman; Article No. 13; pp. 13:1–13:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Theory of computation → Models of computation

Keywords and phrases DFS, BFS, in-place algorithm, space-efficient graph algorithms, logspace

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.13

Related Version A full version of the paper can be found at [21], `https://arxiv.org/abs/1711.09859`.

1 Introduction

Motivated by the rapid growth of huge data set ("big data"), space efficient algorithms are becoming increasingly important than ever before. The proliferation of specialized handheld devices and embedded systems that have a limited supply of memory provide another motivation to consider space efficient algorithms. To design *space-efficient* algorithms in general, several models of computation have been proposed. Among them, the following two computational models have received considerable attention in the literature.

- In the *read-only* memory (ROM) model, we assume that the input is given in a read-only memory. The output of an algorithm is written on a separate write-only memory, and the output can not be read or modified again. In addition to the input and output media, a limited random access workspace is available. Early work on this model was on designing lower bounds [14, 15, 16], for designing algorithms for selection and sorting [26, 38, 43, 49, 50, 52] and problems in computational geometry [6, 9, 12, 25, 30]. Recently there has been interest on space-efficient graph algorithms [7, 10, 11, 22, 23, 24, 37, 44, 45].
- In the *in-place* model, the input elements are given in an array, and the algorithm may use the input array as working space. Hence the algorithm may modify the array during its execution. After the execution, all the input elements should be present in the array (maybe in a permuted order) and the output maybe put in the same array or sent to an output stream. The extra space usage during the execution of the algorithm is limited to $O(\lg n)$ bits. A prominent example of an in-place algorithm is the classic heap-sort. Other than in-place sorting [42], searching [40, 51] and selection [47], many in-place algorithms were designed in areas such as computational geometry [17] and stringology [41].

Apart from these models, researchers have also considered (semi)-streaming models [3, 39, 49] for designing space-efficient algorithms. Very recently the following two new models were introduced in the literature with the same objective.

- Chan et al. [27] introduced the *restore* model which is a more relaxed version of read-only memory (and a restricted version of the in-place model), where the input is allowed to be modified, but at the end of the computation, the input has to be restored to its original form. They also gave space efficient algorithms for selection and sorting on integer arrays in this model. This has motivation, for example, in scenarios where the input (in its original form) is required by some other application.
- Buhrman et al. [18, 19, 46] introduced and studied the *catalytic-space* model where a small amount (typically $O(\lg n)$ bits) of clean space is provided along with some large additional auxiliary space, with the condition that the additional space is initially in an arbitrary, possibly incompressible, state and must be returned to this state when the computation is finished. The input is assumed to be given in ROM. Thus this model can be thought of as having an auxiliary storage that needs to be 'restored' in contrast to the model by Chan et al. [27] where the input array has to be 'restored'. They show various

interesting complexity theoretic consequences in this model and designed space-efficient algorithms in comparison with the ROM model for a few combinatorial problems.

1.1 Previous work on space efficient graph algorithms

Even though these models were introduced in the literature with the aim of designing and/or implementing various algorithms space efficiently, *space efficient graph algorithms* have been designed only in the (semi)-streaming and the ROM model. In the streaming and semi-streaming models, researchers have studied several basic and fundamental algorithmic problems such as connectivity, minimum spanning tree, matching. See [48] for a comprehensive survey in this field. Research on these two models (i.e., streaming and semi-streaming) is relatively new and has been going on for the last decade or so whereas the study in ROM could be traced back to almost 40 years. In fact there is already a rich history of designing space efficient algorithms in the read-only memory model. The complexity class L is the class containing decision problems that can be solved by a deterministic Turing machine using only logarithmic amount of work space for computation. There are several important algorithmic results [31, 34, 35, 36] for this class, the most celebrated being Reingold's method [55] for checking st-reachability in an undirected graph, i.e., to determine if there is a path between two given vertices s and t. NL is the non-deterministic analogue of L and it is known that the st-reachability problem for *directed* graphs is NL-complete (with respect to log space reductions). Using Savitch's algorithm [5], this problem can be solved in $n^{O(\lg n)}$ time using $O(\lg^2 n)$ bits of extra space. Savitch's algorithm is very space efficient but its running time is superpolynomial. Among the deterministic algorithms running in polynomial time for directed st-reachability, the most space efficient algorithm is due to Barnes et al. [13] who gave a slightly sublinear space (using $n/2^{\Theta(\sqrt{\lg n})}$ bits) algorithm for this problem running in polynomial time. We know of no better polynomial time algorithm for this problem with better space bound. Moreover, the space used by this algorithm matches a lower bound on space for solving directed st-reachability on a restricted model of computation called Node Naming Jumping Automata on Graphs (NNJAG's) [28, 33]. This model was introduced especially for the study of directed st-reachability and most of the known sublinear space algorithms for this problem can be implemented on it. Thus, to design any polynomial time ROM algorithm taking space less than $n/2^{\Theta(\sqrt{\lg n})}$ bits requires radically new ideas. Recently there has been some improvement in the space bound for some special classes of graphs like planar and H-minor free graphs [8, 20]. A drawback for all these algorithms using small space i.e., sublinear number of bits, is that their running time is often some polynomial of high degree. This is not surprising as Tompa [57] showed that for directed st-reachability, if the number of bits available is $o(n)$ then some natural algorithmic approaches to the problem require super-polynomial time.

Motivated by these impossibility results from complexity theory and inspired by the practical applications of these fundamental graph algorithms, recently there has been a surge of interest in improving the space complexity of the fundamental graph algorithms without paying too much penalty in the running time i.e., reducing the working space of the classical graph algorithms to $O(n)$ bits with little or no penalty in running time. Thus the goal is to design space-efficient yet reasonably time-efficient graph algorithms on the ROM. Generally most of the classical linear time graph algorithms take $O(n \lg n)$ bits. Recently Asano et al. [7] gave an $O(m \lg n)$ time algorithm using $O(n)$ bits, and another implementation taking $n + o(n)$ bits (using Reingold's or Barnes et al's reachability algorithm) but using high polynomial running time. Later, time bound was improved to $O(m \lg \lg n)$ still using $O(n)$

bits in [37]. For sparse graphs, the time bound is further improved in [10, 22] to optimal $O(m)$ using still $O(n)$ bits of space. Improving on the classical linear time implementation of BFS which uses $O(n \lg n)$ bits of space, recent space efficient algorithms [10, 37, 44] have resulted in a linear time algorithm using $n \lg 3 + o(n)$ bits. We know of no algorithm for BFS using $n + o(n)$ bits and $O(m \lg^c n)$ (or even $O(mn)$) time for some constant c in ROM. The only BFS algorithm taking sublinear space uses $n/2^{\Theta(\sqrt{\lg n})}$ bits [13] and has a high polynomial runtime. A few other space efficient algorithms for fundamental graph problems like checking strong connectivity [37], biconnectivity and performing *st*-numbering [22], recognizing chordal and outerplanar graphs [24, 45] were also designed very recently.

1.2 In-place model for graph algorithms

In order to break these inherent space bound barriers and obtain reasonably time and space efficient graph algorithms, we want to relax the limitations of ROM. And the most natural and obvious candidate in this regard is the classical *in-place* model. Thus our main objective is to initiate a systematic study of efficient in-place (i.e., using $O(\lg n)$ bits of extra space) algorithms for graph problems. To the best of our knowledge, this has not been done in the literature before. Our first goal towards this is to properly define models for in-place graph algorithms. As in the case of the standard in-place model, we need to ensure that the graph (adjacency) structure remains intact throughout the algorithm. Let $G = (V, E)$ be the input graph with $n = |V|$, $m = |E|$, and assume that the vertex set V of G is the set $V = \{1, 2, \cdots, n\}$. To describe these models, we assume that the input graph representation consists of two parts: (i) an array V of length n, where $V[i]$ stores a pointer to the adjacency list of vertex i, and (ii) a list of singly linked lists, where the i-th list consists of a singly linked list containing all the neighbors of vertex i with $V[i]$ pointing to the head of the list. In the ROM model, we assume that both these components cannot be modified. In our relaxed models, we assume that one of these components can be modified in a limited way. This gives rise to two different models which we define next.

Implicit model. The most natural analogue of in-place model allows any two elements in the adjacency list of a vertex to be swapped (in $O(1)$ time assuming that we have access to the nodes storing those elements in the singly linked list). The adjacency "structure" of the representation does not change; only the values stored can be swapped. (One may restrict this further to allow only elements in adjacent nodes to be swapped. Most of our algorithms work with this restriction.) We call it the implicit model inspired by the notion of *implicit data structures* [51].

Rotate model. In this model, we assume that only the pointers stored in the array V can be modified, that too in a limited way - to point to any node in the adjacency list, instead of always pointing to the first node. In space-efficient setting, since we do not have additional space to store a pointer to the beginning of the adjacency list explicitly, we assume that the second component of the graph representation consists of a list of circular linked lists (instead of singly linked lists) – i.e., the last node in the adjacency list of each vertex points to the first node (instead of storing a null pointer). We call the element pointed to by the pointer as the front of the list, and a unit cost rotate operation changes the element pointed to by the pointer to the next element in the list.

Thus the rotate model corresponds to keeping the adjacency lists in read-only memory and allowing (limited) updates on the pointer array that points to these lists. And, the implicit model corresponds to the reverse case, where we keep the pointer array in read-only memory and allow swaps on the adjacency lists/arrays. A third alternative especially for the implicit model is to assume that the input graph is represented as an adjacency array, i.e., adjacency lists are stored as arrays instead of singly linked lists (see [22, 37, 45] for some results using this model); and we allow here that any two elements in the adjacency array can be swapped. In this model, some of our algorithms have improved performance in time.

1.3 Definitions, computational complexity and notations

We study some basic and fundamental graph problems in these models. In what follows we provide the definitions and state the computational complexity of these problems. For the DFS problem, there have been two versions studied in the literature. In the *lexicographically smallest DFS* or *lex-DFS* problem, when DFS looks for an unvisited vertex to visit in an adjacency list, it picks the "first" unvisited vertex where the "first" is with respect to the appearance order in the adjacency list. The resulting DFS tree will be unique. In contrast to lex-DFS, an algorithm that outputs *some* DFS numbering of a given graph, treats an adjacency list as a set, ignoring the order of appearance of vertices in it, and outputs a vertex ordering T such that there exists *some* adjacency ordering R such that T is the DFS numbering with respect to R. We say that such a DFS algorithm performs *general-DFS*. Reif [54] has shown that lex-DFS is P-complete (with respect to log-space reductions) implying that a logspace algorithm for lex-DFS results in the collapse of complexity classes P and L. Anderson et al. [4] have shown that even computing the leftmost root-to-leaf path of the lex-DFS tree is P-complete. For many years, these results seemed to imply that the general-DFS problem, that is, the computation of any DFS tree is also inherently sequential. However, Aggarwal et al. [1, 2] proved that the general-DFS problem can be solved much more efficiently, and it is in RNC. Whether the general-DFS problem is in NC is still open.

As is standard in the design of space-efficient algorithms [10, 37], while working with directed graphs, we assume that the graphs are given as in/out (circular) adjacency lists i.e., for a vertex v, we have the (circular) lists of both in-neighbors and out-neighbors of v. We assume the word RAM model of computation where the machine consists of words of size w in $\Omega(\lg n)$ bits and any logical, arithmetic and bitwise operation involving a constant number of words takes $O(1)$ time. We count space in terms of number of *extra* bits used by the algorithm other than the input, and this quantity is referred as "extra space" and "space" interchangeably throughout the paper. By a path of length d, we mean a simple path on d edges. By $deg(x)$ we mean the degree of the vertex x. In directed graphs, it should be clear from the context whether that denotes out-degree or in-degree. By a BFS/DFS traversal of the input graph G, as in [7, 10, 22, 37, 44], we refer to reporting the vertices of G in the BFS/DFS ordering, i.e., in the order in which the vertices are visited for the first time.

1.4 Our Results

Depth-first Search. In the rotate model, we show the following in Sections 2.1, 2.2 and 2.3.

▶ **Theorem 1.** *Let G be a directed or an undirected graph, and $\ell \leq n$ be the maximum depth of the DFS tree starting at a source vertex s. Then in the* rotate *model, the vertices of G can be output in*

(a) *the lex-DFS order in $O(m+n)$ time using $n \lg 3 + O(\lg^2 n)$ bits of extra space,*
(b) *a general-DFS order in $O(m+n)$ time using $n + O(\lg n)$ bits of extra space, and*
(c) *a general-DFS order in $O(m^2/n + m\ell)$ time for an undirected graph and in $O(m(n+\ell^2))$ time for directed graphs using $O(\lg n)$ bits of extra space. For this algorithm, we assume that s can reach all other vertices.*

In the implicit model, we obtain polynomial time implementations for lex-DFS and general-DFS using $O(\lg n)$ bits of extra space. For lex-DFS, this is conjectured to be unlikely in ROM as the problem is P-complete [54]. In particular, we show the following in Section 4.

▶ **Theorem 2.** *Let G be a directed or an undirected graph with a source vertex s and $\ell \leq n$ be the maximum depth of any DFS tree starting at s that can reach all other vertices. Then in the* implicit *model, using $O(\lg n)$ bits of extra space, the vertices of G can be output in*

(a) *the lex-DFS order in $O(m^3/n^2 + \ell m^2/n)$ time if G is given in adjacency list and in $O(m^2 \lg n/n)$ time if G is given in adjacency array for undirected graphs. For directed graphs our algorithm takes $O(m^2(n + \ell^2)/n)$ time if G is given in adjacency list and $O(m \lg n(n + \ell^2))$ time if G is given in adjacency array;*

(b) *a general-DFS traversal order in $O(m^2/n)$ time if the input graph G is given in an adjacency list and in $O(m^2(\lg n)/n + m\ell \lg n))$ time if it is given in an adjacency array.*

In sharp contrast, for space efficient algorithms for DFS in ROM, the landscape looks markedly different. To the best of our knowledge, there are no DFS algorithms in general graphs in ROM that use $O(n^{1-\epsilon})$ bits. In fact, an implementation of DFS taking cn bits for $c < 1$ has been proposed as an open problem in [7].

Breadth-first Search. In the rotate model, we show the following.

▶ **Theorem 3.** (♠)[3] *Let G be a directed or an undirected graph, and $\ell \leq n$ be the depth of the BFS tree starting at the source vertex s. Then in the* rotate *model, the vertices of G can be output in a BFS order in*

(a) *$O(m + n\ell^2)$ time using $n + O(\lg n)$ bits of extra space, and*

(b) *$O(m\ell + n\ell^2)$ time using $O(\lg n)$ bits of extra space. Here we assume that the source vertex can reach all other vertices.*

In the implicit model, we can match the runtime of BFS from rotate model, and do better in some special classes of graphs. In particular, we show the following.

▶ **Theorem 4.** (♠) *Let G be a directed or an undirected graph with a source vertex that can reach all other vertices by a distance of at most $\ell \leq n$. Then in the* implicit *model, using $O(\lg n)$ bits of extra space, the vertices of G can be output in a BFS order in*

(a) *$O(m + n\ell^2)$ time;*

(b) *the runtime can be improved to $O(m + n\ell)$ time if there are no degree 2 vertices;*

(c) *the runtime can be improved to $O(m)$ if the degree of every vertex is at least $2 \lg n + 3$.*

Similar to DFS, to the best of our knowledge, there are no polynomial time BFS algorithms in ROM that use even $O(n^{1-\epsilon})$ bits. On the other hand, we don't hope to have a BFS algorithm (for both undirected and directed graphs) using $O(\lg n)$ bits of extra space in ROM as the problem is NL-complete [5].

Minimum Spanning Tree (MST). We also study the problem of reporting the edges of a MST of a given undirected connected graph G and we show the following.

[3] Proofs of results marked with (♠) appear in the full version [21].

▶ **Theorem 5.** (♠) *A minimum spanning tree of a given undirected weighted graph G can be found using $O(\lg n)$ bits of extra space and in*
(a) *$O(mn)$ time in the* rotate *model,*
(b) *$O(mn^2)$ time in the* implicit *model if G is given in an adjacency list, and*
(c) *$O(mn \lg n)$ time in the* implicit *model when G is represented in an adjacency array.*

Note that by the results of [53, 55], we already know log-space algorithms for MST in ROM but again the drawback of those algorithms is their large time complexity. On the other hand, our algorithms have relatively small polynomial running time, simplicity, making it an appealing choice in applications with strict space constraints.

1.5 Techniques

Our implementations follow (variations of) the classical algorithms for BFS and DFS that use three colors (white, gray and black), but avoid the use of stack (for DFS) and queue (for BFS). In the rotate model, we first observe that in the usual search algorithms one can dispense with the extra data structure space of pointers maintaining the search tree (while retaining the linear number of bits and a single bit per vertex in place of the full unvisited/visited/explored array) simply by rotating each circular adjacency lists to move the parent or a (typically the currently explored) child to the beginning of the list to help navigate through the tree during the forward or the backtracking step, i.e. by changing the pointer from the vertex to the list of its adjacencies by one node at a time. This retains the basic efficiency of the search strategies. The nice part of this strategy is that the total number of rotations also can be bounded. To reduce the extra space from linear to logarithmic, it is noted that one can follow the vertices based on the current rotations at each vertex to determine the visited status of a vertex, i.e. these algorithms use the rotate operation in a non-trivial way to move elements within the lists to determine the color of the vertices as well. However, the drawback is that to do so could require moving up (or down) the full height of the implicit search tree. This yields polynomial rather than (near-)linear time algorithms.

In the implicit model, we use the classical *bit encoding* trick used in the development of the implicit data structures [51]. We encode one (or two) bit(s) using a sequence of two (or three respectively) distinct numbers. To encode a single bit b using two distinct values x and y with $x < y$, we store the sequence x, y if $b = 0$, and y, x otherwise. Similarly, permuting three distinct values x, y, z with $x < y < z$, we can represent six combinations. We can choose any of the four combinations to represent up to 4 colors (i.e. two bits). Generalizing this further, we can encode a pointer taking $\lg n$ bits using $2 \lg n$ distinct elements where reading or updating a bit takes constant time, and reading or updating a pointer takes $O(\lg n)$ time. This also is the reason for the requirement of vertices with degree at least 3 or $2 \lg n + 3$ for faster algorithms, which will become clear in the description of the algorithms.

1.6 Consequences of our BFS and DFS results

There are many interesting and surprising consequences of our results for BFS and DFS in both the rotate and implicit model. In what follows, we mention a few of them. See the full version [21] for the complete discussion on the consequences of our results.

- For *directed st-reachability*, as mentioned previously, the most space efficient polynomial time algorithm [13] uses $n/2^{\Theta(\sqrt{\lg n})}$ bits. In sharp contrast, we obtain efficient (timewise) log-space algorithms for this problem in both the rotate and implicit models (as a corollary of our directed graph DFS/BFS results). In terms of workspace this is exponentially better than the best known polynomial time algorithm [13] for this problem in ROM. For us, this provides one of the main motivations to study this model. A somewhat incomparable result obtained recently by Buhrman et al. [18, 46] where they designed an algorithm for *directed st-reachability* on catalytic Turing machines in space $O(\lg n)$ with catalytic space $O(n^2 \lg n)$ and time $O(n^9)$.

- Problems like *directed st-reachability* [5], *distance* [56] which asks whether a given G (directed, undirected or even directed acyclic) contains a path of length at most k from s to t, are NL-complete i.e., no deterministic log-space algorithm is known. But in both the rotate and implicit models, we design log-space algorithms for them. Assuming $\mathsf{L} \neq \mathsf{NL}$, these results show that probably both our models with log-space are stronger than NL.
- The lex-DFS problem (both in undirected and directed graphs) is P-complete [54], and thus polylogarithmic space algorithms are unlikely to exist in the ROM model. But we show an $O(\lg n)$ space algorithm in the implicit model for lex-DFS. This implies that, probably the implicit model is even more powerful than the rotate model. It could even be possible that every problem in P can be computed using log-space in the implicit model. A result of somewhat similar flavor is obtained recently Buhrman et al. [18, 46] where they showed that any function in TC^1 can be computed using catalytic log-space, i.e., $\mathsf{TC}^1 \subseteq \mathsf{CSPACE}(\lg n)$. Note that TC^1 contains L, NL and even other classes that are conjectured to be different from L.
- For a large number of NP-hard graph problems, the best algorithms in ROM run in exponential time and polynomial space. We show that using just logarithmic amount of space, albeit using exponential time, we can design algorithms for those NP-hard problems in both of our models under some restrictions. This gives an exponential improvement over the ROM space bounds for these problems. In constrast, no NP-hard problem can be solved in the ROM model using $O(\lg n)$ bits unless P=NP. We discuss the details in the full version [21].

2 DFS algorithms in the rotate model

We first describe our space-efficient algorithms for DFS in the rotate model proving Theorem 1.

2.1 Proof of Theorem 1(a) for undirected graphs

We begin by describing our algorithm for undirected graphs, and later mention the changes required for directed graphs. In the normal exploration of DFS (see for example, Cormen et al. [29]) we use three colors. Every vertex v is white initially while it has not been discovered yet, becomes gray when DFS discovers v for the first time, and is colored black when it is finished i.e., all its neighbors have been explored completely.

We maintain a color array C of length n that stores the color of each vertex at any point in the algorithm. In the rest of the paper, when we say we scan the adjacency list of some vertex v, what we mean is, we create a temporary pointer pointing to the current first element of the list and move this temporary pointer until we find the desired element. Once we get that element we actually rotate the list so that the desired element now is at the front of the list. We start DFS at the starting vertex, say s, changing its color from white to gray in the color array C. Then we scan the adjacency list of s to find the first white neighbor, say w. We keep rotating the list to bring w to the front of s's adjacency list (as the one pointed to by the head $V[s]$), color w gray in the color array C and proceed to the next step (i.e. to explore w's adjacency list). This is the first *forward* step of the algorithm. In general, at any step during the execution of the algorithm, whenever we arrive at a gray vertex u

(either from one of u's black children or when u's color is changed from white to gray in the current step), we scan u's adjacency list to find the first white vertex. (i) If we find such a vertex, say v, then we rotate u's list to make v as the first element, and change the color of v to gray. (ii) If we do not find any white vertex, then we change the color of u to black, and *backtrack* to its parent. To identify u's parent, we use the following lemma.

▶ **Lemma 6.** (♠) *Suppose w is a node that just became* black. *Then its parent p is the unique vertex in w's list which is (a)* gray *and (b) whose current list has w in the first position.*

So, the parent can be found by scanning the w's list, to find a neighbor p that is colored gray such that the first element in p's list is u. This completes the description of the backtracking step. Once we backtrack to p, we find the next white vertex (as in the forward step) and continue until all the vertices of G are explored. Other than some constant number of variables, clearly the space usage is only for storing the color array C. Since C is of length n where each element has 3 possible values, C can be encoded using $n \lg 3 + O(\lg^2 n)$ bits, so that the i-th element in C can be read and updated in $O(1)$ time [32]. So overall space required is $n \lg 3 + O(\lg^2 n)$ bits. It is easy to see that at most two full rotations of each of the list may happen during the execution of the algorithm (first one to explore all the white neighbors and the second one to determine that there are no more white neighbors) resulting in a linear time lex-DFS algorithm. We discuss the lex-DFS algorithm for the directed graphs in the full version of the paper [21].

2.2 Proof of Theorem 1(b) for undirected graphs

To improve the space further, we replace the color array C with a bit array $visited[1, \dots, n]$ which stores a 0 for an unvisited vertex (white), and a 1 for a visited vertex (gray or black). First we need a test similar to that in the statement of Lemma 6 without the distinction of gray and black vertices to find the parent of a node. Due to the invariant we have maintained, every internal vertex of the DFS tree will point to (i.e. have as first element in its list) its current last child. So the nodes that could potentially have node w in its first position are its parent, and any of its children which is a leaf. Hence we modify the forward step in the following way.

Whenever we visit an unvisited vertex v for the first time from another vertex u (hence, u is the parent of v in the DFS tree and u's list has v in the first position), we, as before, mark v as visited and in addition to that, we rotate v's list to bring u to the front (during this rotation, we do not mark any intermediate nodes as visited). Then we continue as before (by finding the first unvisited vertex and bringing it to the front) in the forward step. Now the following invariants are easy to see and are useful.

Invariants: During the exploration of DFS, in the (partial) DFS tree
1. any internal vertex has the first element in its list as its current last child; and
2. for any leaf vertex of the DFS tree, the first element in its list is its parent.

The first invariant is easy to see as we always keep the current explored vertex (child) as the first element in the list. For leaves, the first time we encounter them, we make its parent as the first element in the forward direction. Then we discover that it has no unvisited vertices in its list, and so we make a full rotation and bring the parent to the front again. The following lemma provides a test to find the parent of a node.

▶ **Lemma 7.** (♠) *Let w be a node that has just become* black. *Then its parent p is the* ***first*** *vertex x in w's list whose current adjacency list has w in the first position.*

Once we backtrack to p, we find the next white vertex, and continue until all the vertices of G are explored. Overall this procedure takes linear time. As we rotate the list to bring the parent of a node, before exploring its white neighbors, we are not guaranteed to explore the first white vertex in its original list, and hence we loose the lexicographic property. We provide our DFS algorithm for directed graphs in the full version [21].

2.3 Proof of Theorem 1(c) for undirected graphs

Now to decrease the space to $O(\lg n)$, we dispense with the color/visited array, and give tests to determine white, gray and black vertices. For now, assume that we can determine the color of a vertex. The forward step is almost the same as before except performing the update in the color array. I.e., whenever we visit a white vertex v for the first time from another vertex u (hence u is the parent of v), we rotate v's list to bring u to the front. Then we continue to find the first white vertex to explore. We maintain the following invariants. (i) any gray vertex has the first element in its list as its last child in the (partial) DFS tree; (ii) any black vertex has its parent as the first element in its list. We also store the depth of the current node in a variable d, which is incremented by 1 every time we discover a white vertex and decremented by 1 whenever we backtrack. We maintain the maximum depth the DFS has attained using a variable max. At a generic step during the execution of the algorithm, assume that we are at a vertex x's list, let p be x's parent and let y be a vertex in x's list. We need to determine the color of y and continue the DFS based on the color of y. We use the following characterization.

▶ **Lemma 8.** (♠) *Suppose the DFS has explored starting from a source vertex s, up to a vertex x at level d. Let p be x's parent. Note that both s and x are* gray *in the normal coloring procedure. Let max be the maximum level of any vertex in the partial DFS exploration. Let y be a vertex in x's list. Then,*

1. *y is* gray *(i.e., (x, y) is a back edge, and y is an ancestor of x) if and only if we can reach y from s by following through the* gray *child (which is at the front of a* gray *node's list) path in at most d steps.*
2. *y is* black *(i.e., (x, y) is a back edge, and x is an ancestor of y) if and only if*
   - *there is a path P of length at most $(max - d)$ from y to x (obtained by following through the first elements of the lists of every vertex in the path, starting from y), and*
   - *let z be the node before x in the path P. The node z appears after p in x's list.*
3. *y is* white *if y is not* gray *or* black.

Now, if we use the above claim to test for colors of vertices, testing for gray takes at most d steps. Testing for black takes at most $(max - d)$ steps to find the path, and at most $deg(x)$ steps to determine whether p appears before. Thus for each vertex in x's list, we spend time proportional to $max + deg(x)$. So, the overall runtime of the algorithm is $\sum_{v \in V} deg(v)(deg(v) + \ell) = O(m^2/n + m\ell)$, where ℓ is the maximum depth of DFS tree. Maintaining the invariants for the gray and black vertices are also straightforward. We provide the details of our log-space algorithm for directed graphs in the full version [21].

3 Simulation of algorithms for rotate model in the implicit model

The following result captures the overhead incurred while simulating any rotate model algorithm in the implicit model.

▶ **Theorem 9.** (♠) *Let D be the maximum degree of G. Then any algorithm running in $t(m,n)$ time in the* rotate *model can be simulated in the* implicit *model in (i) $O(D \cdot t(m,n))$ time when G is given in an adjacency list, and (ii) $O(\lg D \cdot t(m,n))$ time when G is given in an adjacency array. Furthermore, let $r_v(m,n)$ denote the number of rotations made in v's (whose degree is d_v) list, and $f(m,n)$ be the remaining number of operations. Then any algorithm running in $t(m,n) = f(m,n) + \sum_{v \in V} r_v(m,n)$ time in the* rotate *model can be simulated in the* implicit *model in (i) $O(f(m,n) + \sum_{v \in V} r_v(m,n) \cdot d_v)$ time when G is given in an adjacency list, and (ii) $O(f(m,n) + \sum_{v \in V} r_v(m,n) \lg d_v)$ time when G is given in an adjacency array.*

Most of our algorithms in the implicit model use these simulations often with some enhancements and tricks to obtain better running time bounds for some specific problems.

4 DFS algorithms in the implicit model – proof of Theorem 2

To obtain a lex-DFS algorithm, we implement the $O(\lg n)$-bit DFS algorithm in the rotate model, described in Section 2.3, with a simple modification. First, note that in this algorithm (in the rotate model), we bring the parent of a vertex to the front of its adjacency list (by performing rotations) when we visit a vertex for the first time. Subsequently, we explore the remaining neighbors of the vertex in the left-to-right order. Thus, for each vertex, if its parent in the DFS were at the beginning of its adjacency list, then this algorithm would result in a lex-DFS algorithm. Now, to implement this algorithm in the implicit model, whenever we need to bring the parent to the front, we simply bring it to the front without changing the order of the other neighbors. Subsequently, we simulate each rotation by moving all the elements in the adjacency list circularly. As mentioned in Section 3, this results in an algorithm whose running time is $O(\sum_{v \in V} d_v(d_v + \ell) \cdot d_v) = O(m^3/n^2 + \ell m^2/n)$ if the graph is given in an adjancecy list and in $O(\sum_{v \in V} d_v(d_v + \ell) \cdot \lg d_v) = O(m^2(\lg n)/n + m\ell \lg n))$ when the graph is given in the form of an adjacency array. This proves Theorem 2(a) for undirected graphs. The results for the directed case follow from simulating the corresponding results for the directed graphs.

To prove the result mentioned in Theorem 2(b), we implement the linear-time DFS algorithm of Theorem 1 for the rotate model that uses $n + O(\lg n)$ bits. This results in an algorithm that runs in $O(\sum_{v \in V} d_v^2 + n) = O(m^2/n)$ time (or in $O(\sum_{v \in V} d_v \lg d_v + n) = O(m \lg m + n)$ time, when the graph is given as an adjacency array representation), using $n + O(\lg n)$ bits. We reduce the space usage of the algorithm to $O(\lg n)$ bits by encoding the visited/unvisited bit for each vertex with degree at least 2 within its adjacency list (and not maintaining this bit for degree-1 vertices). We describe the details below.

Whenever a node is visited for the first time in the algorithm for the rotated list model, we bring its parent to the front of its adjacency list. In the remaining part of the algorithm, we process each of its other adjacent vertices while rotating the adjacency list, untill the parent comes to the front again. Thus, for each vertex v with degree d_v, we need to rotate v's adjacency list $O(d_v)$ times. In the implicit model, we also bring the parent to the front when a vertex is visited for the first time, for any vertex with degree at least 3. We use the second and third elements in the adjacency list to encode the visited/unvisited bit. But instead of rotating the adjacency list circularly, we simply scan through the adjacency list from left to right everytime we need to find the next unvisited vertex in its adjacency list. This requires $O(d_v)$ time for a vertex v with degree d_v. We show how to handle vertices with degree at most 2 separately.

As before, we can deal with the degree-1 vertices without encoding visited/unvisited bit as we encounter those vertices only once during the algorithm. For degree-2 vertices, we initially (at preprocessing stage) encode the bit 0 using the two elements in their adjacency arrays – to indicate that they are unvisited. When a degree-2 vertex is visited for the first time from a neighbor x, we move to its other neighbor – continuing the process as long as we encounter degree-2 vertices until we reach a vertex y with degree at least 3. If y is already visited, then we output the path consisting of all the degree-2 vertices and backtrack to x. If y is not visited yet, then we output the path up to y, and continue the search from y, and after marking y as visited. In both cases, we also mark all the degree-2 nodes as visited (by swapping the two elements in each of their adjacency arrays).

During the preprocessing, for each vertex with degree at least 3, we ensure that the second and third elements in its adjacency list encode the bit 0 (to mark it unvisited). We maintain the invariant that for any vertex with degree at least 3, as long as it is not visited, the second and third elements in its adjacency array encode the bit 0; and after the vertex is visited, its parent (in the DFS tree) is at the front of its adjacency array, and the second and third elements in its adjacency array encode the bit 1. Thus, when we visit a node v with degree at least 3 for the first time, we bring its parent to the front, and then swap the second and third elements in the adjacency list, if needed, to mark it as visited. The total running time of this algorithm is bounded by $\sum_{v \in V} d_v^2 = O(m^2/n)$.

We can implement the above DFS algorithm even faster when the input graph is given in an adjacency array representation. We deal with vertices with degree at most 2 exactly as before. For a vertex v with degree at least 3, we bring its parent to the front and swap the second and third elements to mark the node as visited (as before) whenever v is visited for the first time. We then sort the remaining elements, if any, in the adjacency array, in-place (using the linear-time in-place radix sort algorithm [42]), and implement the rotations on the remaining part of the array as described in Section 3. The total running time of this algorithm is bounded by $\sum_{v \in V} d_v \lg d_v = O(m \lg m + n)$. This completes the proof of Theorem 2(b).

5 Concluding remarks

Our initial motivation was to get around the limitations of ROM to obtain a reasonable model for graphs in which we can obtain space efficient algorithms. We achieved that by introducing two new frameworks and obtained efficient (of the order of $O(n^3 \lg n)$) algorithms using $O(\lg n)$ bits of space for fundamental graph search procedures. We also discussed various applications of our DFS/BFS results, and it is not surprising that many corollaries would follow as they are the backbone of many algorithms. We showed that some of these results also translate to improved space efficient algorithms in ROM (by simulating the rotate model algorithms in ROM with a pointer per list). With some effort, we can obtain log space algorithm for MST. These results can be contrasted with the state of the art results in ROM that take almost linear bits for some of these problems other than having large runtime bounds. We believe that our work is the first step towards designing efficient in-place graph algorithms and it will inspire further investigation into designing such algorithms for other graph problems. One future direction would also be to improve the running times of our algorithms. As in the case of most of the earlier space-efficient graph algorithms, we only consider adjacency list and array representation. It's not clear how to define in-place model for adjacency matrix. Another challenging direction would be to design efficient algorithms that also restore the initial input representation at the end of the execution of the algorithm.

Surprisingly we could design log-space algorithm for some P-complete problems, and so it is important to understand the power of our models. Towards that we discovered that we can even obtain log-space algorithms for some NP-hard graph problems. More specifically, we defined *graph subset problems* and obtained log-space exponential time algorithms for problems belonging to this class in [21]. One interesting future direction would be to determine the exact computational power of these models along with exploring the horizon of interesting complexity theoretic consequences of problems in these models. Unlike the ROM model, it's not clear how one can define an in-place model which is closed under composition. We leave this as a challenging open problem.

References

1 A. Aggarwal and R. J. Anderson. A random NC algorithm for depth first search. *Combinatorica*, 8(1):1–12, 1988. doi:10.1007/BF02122548.

2 A. Aggarwal, R. J. Anderson, and M. Kao. Parallel depth-first search in general directed graphs. *SIAM J. Comput.*, 19(2):397–409, 1990. doi:10.1137/0219025.

3 N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. doi:10.1006/jcss.1997.1545.

4 R. J. Anderson and E. W. Mayr. Parallelism and the maximal path problem. *Inf. Process. Lett.*, 24(2):121–126, 1987. doi:10.1016/0020-0190(87)90105-0.

5 S. Arora and B. Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

6 T. Asano, K. Buchin, M. Buchin, M.Korman, W. Mulzer, G. Rote, and A. Schulz. Reprint of: Memory-constrained algorithms for simple polygons. *Comput. Geom.*, 47(3):469–479, 2014. doi:10.1016/j.comgeo.2013.11.004.

7 T. Asano, T. Izumi, M. Kiyomi, M. Konagaya, H. Ono, Y. Otachi, P. Schweitzer, J. Tarui, and R. Uehara. Depth-first search using $O(n)$ bits. In *25th ISAAC*, pages 553–564, 2014.

8 T. Asano, D. G. Kirkpatrick, K. Nakagawa, and O. Watanabe. $\tilde{O}(\sqrt{n})$-space and polynomial-time algorithm for planar directed graph reachability. In *39th MFCS LNCS 8634*, pages 45–56, 2014. doi:10.1007/978-3-662-44465-8_5.

9 T. Asano, W. Mulzer, G. Rote, and Y. Wang. Constant-work-space algorithms for geometric problems. *JoCG*, 2(1):46–68, 2011. URL: http://jocg.org/index.php/jocg/article/view/30.

10 N. Banerjee, S. Chakraborty, and V. Raman. Improved space efficient algorithms for BFS, DFS and applications. In *22nd COCOON*, 2016. URL: http://arxiv.org/abs/1606.04718.

11 N. Banerjee, S. Chakraborty, V. Raman, S. Roy, and S. Saurabh. Time-space tradeoffs for dynamic programming in trees and bounded treewidth graphs. In *21st COCOON*, volume 9198, pages 349–360. springer, LNCS, 2015.

12 L. Barba, M. Korman, S. Langerman, K. Sadakane, and R. I. Silveira. Space-time trade-offs for stack-based algorithms. *Algorithmica*, 72(4):1097–1129, 2015. doi:10.1007/s00453-014-9893-5.

13 G. Barnes, J. Buss, W. Ruzzo, and B. Schieber. A sublinear space, polynomial time algorithm for directed *s-t* connectivity. *SIAM J. Comput.*, 27(5):1273–1282, 1998. doi:10.1137/S0097539793283151.

14 Paul Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM J. Comput.*, 20(2):270–277, 1991. doi:10.1137/0220017.

15 A. Borodin and S. A. Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM J. Comput.*, 11(2):287–297, 1982. doi:10.1137/0211022.

16 A. Borodin, M. J. Fischer, D. G. Kirkpatrick, N. A. Lynch, and M. Tompa. A time-space tradeoff for sorting on non-oblivious machines. *J. Comput. Syst. Sci.*, 22(3):351–364, 1981. doi:10.1016/0022-0000(81)90037-4.

17 H. Brönnimann, T. M. Chan, and E. Y. Chen. Towards in-place geometric algorithms and data structures. In *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*, pages 239–246, 2004. doi:10.1145/997817.997854.

18 H. Buhrman, R. Cleve, M. Koucký, B. Loff, and F. Speelman. Computing with a full memory: catalytic space. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 857–866, 2014. doi:10.1145/2591796.2591874.

19 H. Buhrman, M.l Koucký, B. Loff, and F. Speelman. Catalytic space: Non-determinism and hierarchy. In *33rd STACS 2016, February 17-20, 2016, Orléans, France*, pages 24:1–24:13, 2016. doi:10.4230/LIPIcs.STACS.2016.24.

20 D. Chakraborty, A. Pavan, R. Tewari, N. V. Vinodchandran, and L. Yang. New time-space upperbounds for directed reachability in high-genus and h-minor-free graphs. In *FSTTCS*, pages 585–595, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.585.

21 S. Chakraborty, A. Mukherjee, V. Raman, and S. R. Satti. Frameworks for designing in-place graph algorithms. *CoRR*, abs/1711.09859, 2017. arXiv:1711.09859.

22 S. Chakraborty, V. Raman, and S. R. Satti. Biconnectivity, chain decomposition and st-numbering using O(n) bits. In *27th ISAAC*, pages 22:1–22:13, 2016. doi:10.4230/LIPIcs.ISAAC.2016.22.

23 S. Chakraborty, V. Raman, and S. R. Satti. Biconnectivity, st-numbering and other applications of DFS using O(n) bits. *J. Comput. Syst. Sci.*, 90:63–79, 2017. doi:10.1016/j.jcss.2017.06.006.

24 S. Chakraborty and S. R. Satti. Space-efficient algorithms for maximum cardinality search, stack bfs, queue BFS and applications. In *Computing and Combinatorics - 23rd International Conference, COCOON 2017, Hong Kong, China, August 3-5, 2017, Proceedings*, pages 87–98, 2017. doi:10.1007/978-3-319-62389-4_8.

25 T. M. Chan and E. Y. Chen. Multi-pass geometric algorithms. *Discrete & Computational Geometry*, 37(1):79–102, 2007. doi:10.1007/s00454-006-1275-6.

26 T. M. Chan, J. I. Munro, and V. Raman. Faster, space-efficient selection algorithms in read-only memory for integers. In *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, pages 405–412, 2013. doi:10.1007/978-3-642-45030-3_38.

27 T. M. Chan, J. I. Munro, and V. Raman. Selection and sorting in the "restore" model. In *25th-SODA*, pages 995–1004, 2014. doi:10.1137/1.9781611973402.74.

28 S. A. Cook and C. Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM J. Comput.*, 9(3):636–652, 1980. doi:10.1137/0209048.

29 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

30 O. Darwish and A. Elmasry. Optimal time-space tradeoff for the 2d convex-hull problem. In *22th ESA*, pages 284–295, 2014. doi:10.1007/978-3-662-44777-2_24.

31 S. Datta, N. Limaye, P. Nimbhorkar, T. Thierauf, and F. Wagner. Planar graph isomorphism is in log-space. In *24th CCC*, pages 203–214, 2009. doi:10.1109/CCC.2009.16.

32 Y. Dodis, M. Patrascu, and M. Thorup. Changing base without losing space. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*, pages 593–602, 2010. doi:10.1145/1806689.1806770.

33 J. Edmonds, C. K. Poon, and D. Achlioptas. Tight lower bounds for st-connectivity on the NNJAG model. *SIAM J. Comput.*, 28(6):2257–2284, 1999. doi:10.1137/S0097539795295948.

34 M. Elberfeld, A. Jakoby, and T. Tantau. Logspace versions of the theorems of bodlaender and courcelle. In *51th FOCS*, pages 143–152, 2010. doi:10.1109/FOCS.2010.21.

35 M. Elberfeld and K. Kawarabayashi. Embedding and canonizing graphs of bounded genus in logspace. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 383–392, 2014. doi:10.1145/2591796.2591865.

36 M. Elberfeld and P. Schweitzer. Canonizing graphs of bounded tree width in logspace. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 32:1–32:14, 2016. doi:10.4230/LIPIcs.STACS.2016.32.

37 A. Elmasry, T. Hagerup, and F. Kammer. Space-efficient basic graph algorithms. In *32nd STACS*, pages 288–301, 2015. doi:10.4230/LIPIcs.STACS.2015.288.

38 A. Elmasry, D. D. Juhl, J. Katajainen, and S. R. Satti. Selection from read-only memory with limited workspace. *Theor. Comput. Sci.*, 554:64–73, 2014. doi:10.1016/j.tcs.2014.06.012.

39 J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005. doi:10.1016/j.tcs.2005.09.013.

40 G. Franceschini and J. Ian Munro. Implicit dictionaries with $O(1)$ modifications per update and fast search. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 404–413, 2006.

41 G. Franceschini and S. Muthukrishnan. In-place suffix sorting. In *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, pages 533–545, 2007. doi:10.1007/978-3-540-73420-8_47.

42 G. Franceschini, S. Muthukrishnan, and M. Patrascu. Radix sorting with no extra space. In *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 194–205, 2007. doi:10.1007/978-3-540-75520-3_19.

43 G. N. Frederickson. Upper bounds for time-space trade-offs in sorting and selection. *J. Comput. Syst. Sci.*, 34(1):19–26, 1987. doi:10.1016/0022-0000(87)90002-X.

44 T. Hagerup and F. Kammer. Succinct choice dictionaries. *CoRR*, abs/1604.06058, 2016. URL: http://arxiv.org/abs/1604.06058, arXiv:1604.06058.

45 F. Kammer, D. Kratsch, and M. Laudahn. Space-efficient biconnected components and recognition of outerplanar graphs. In *41st MFCS*, 2016.

46 M. Koucký. Catalytic computation. *Bulletin of the EATCS*, 118, 2016. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/400.

47 T. W. Lai and D. Wood. Implicit selection. In *SWAT 88, 1st Scandinavian Workshop on Algorithm Theory, Halmstad, Sweden, July 5-8, 1988, Proceedings*, pages 14–23, 1988. doi:10.1007/3-540-19487-8_2.

48 A. McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43(1):9–20, 2014. doi:10.1145/2627692.2627694.

49 J. I. Munro and M. Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980. doi:10.1016/0304-3975(80)90061-4.

50 J. I. Munro and V. Raman. Selection from read-only memory and sorting with minimum data movement. *Theor. Comput. Sci.*, 165(2):311–323, 1996. doi:10.1016/0304-3975(95)00225-1.

51 J. Ian Munro. An implicit data structure supporting insertion, deletion, and search in $O(\log^2 n)$ time. *J. Comput. Syst. Sci.*, 33(1):66–74, 1986. doi:10.1016/0022-0000(86)90043-7.

52 J. Pagter and T. Rauhe. Optimal time-space trade-offs for sorting. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 264–268, 1998. doi:10.1109/SFCS.1998.743455.

53 J. H. Reif. Symmetric complementation. *J. ACM*, 31(2):401–421, 1984. doi:10.1145/62.322436.

54 J. H. Reif. Depth-first search is inherently sequential. *Inf. Process. Lett.*, 20(5):229–234, 1985. doi:10.1016/0020-0190(85)90024-9.

55 O. Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008. doi:10.1145/1391289.1391291.

56 T. Tantau. Logspace optimization problems and their approximability properties. *Theory Comput. Syst.*, 41(2):327–350, 2007. doi:10.1007/s00224-007-2011-1.

57 M. Tompa. Two familiar transitive closure algorithms which admit no polynomial time, sublinear space implementations. *SIAM J. Comput.*, 11(1):130–137, 1982. doi:10.1137/0211010.

Self-Assembly of Any Shape with Constant Tile Types using High Temperature

Cameron Chalk[1]
Department of Electrical and Computer Engineering, University of Texas - Austin, Austin, TX, USA
ctchalk@utexas.edu

Austin Luchsinger[2]
Department of Computer Science, University of Texas - Rio Grande Valley, Edinburg, TX, USA
austin.luchsinger01@utrgv.edu

Robert Schweller[3]
Department of Computer Science, University of Texas - Rio Grande Valley, Edinburg, TX, USA
robert.schweller@utrgv.edu

Tim Wylie[4]
Department of Computer Science, University of Texas - Rio Grande Valley, Edinburg, TX, USA
timothy.wylie@utrgv.edu

Abstract

Inspired by nature and motivated by a lack of top-down tools for precise nanoscale manufacture, self-assembly is a bottom-up process where simple, unorganized components autonomously combine to form larger more complex structures. Such systems hide rich algorithmic properties – notably, Turing universality – and a self-assembly system can be seen as both the object to be manufactured as well as the machine controlling the manufacturing process. Thus, a benchmark problem in self-assembly is the unique assembly of shapes: to design a set of simple agents which, based on aggregation rules and random movement, self-assemble into a particular shape and nothing else. We use a popular model of self-assembly, the 2-handed or hierarchical tile assembly model, and allow the existence of repulsive forces, which is a well-studied variant. The technique utilizes a finely-tuned temperature (the minimum required affinity required for aggregation of separate complexes).

We show that calibrating the temperature and the strength of the aggregation between the tiles, one can encode the shape to be assembled without increasing the number of distinct tile types. Precisely, we show one tile set for which the following holds: for any finite connected shape S, there exists a setting of binding strengths between tiles and a temperature under which the system uniquely assembles S at some scale factor. Our tile system only uses one repulsive glue type and the system is growth-only (it produces no unstable assemblies). The best previous unique shape assembly results in tile assembly models use $\mathcal{O}(\frac{K(S)}{\log K(S)})$ distinct tile types, where $K(S)$ is the Kolmogorov (descriptional) complexity of the shape S.

2012 ACM Subject Classification Theory of computation → Self-organization

Keywords and phrases self-assembly, molecular computing, tiling, tile, shapes

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.14

[1] This author's research was supported in part by National Science Foundation Grants CCF-1618895 and CCF-1652824.
[2] This author's research was supported in part by National Science Foundation Grant CCF-1817602.
[3] This author's research was supported in part by National Science Foundation Grant CCF-1817602.
[4] This author's research was supported in part by National Science Foundation Grant CCF-1817602.

© Cameron Chalk, Austin Luchsinger, Robert Schweller, and Timothy Wylie;
licensed under Creative Commons License CC-BY

1 Introduction

Due to the limited tool set for precise fabrication at the nanoscale, the bottom-up approach of self-assembly is an attractive area of research. Such bottom-up approaches, such as *DNA origami* [17], allow for the assembly of nanoscale materials with detailed, precisely designed shapes and patterns. Abstract self-assembly models are used to predict the behavior of systems wherein simple, separate entities form larger complex structures based on a simple rule set for movement and/or attachment using only local interactions and no global leader. Such systems include swarm robotics and molecular self-assembly, particularly self-assembling nucleic acid structures such as *DNA tiles* [9].

A common benchmark in such models, which aims at the manufacture of precise nanoscale structures, is the self-assembly of *shapes*. In our case, a shape is defined simply as a finite, connected subset of $\mathbb{Z}^2$. The model studied herein is the *two-handed tile assembly model* (also called the *hierarchical tile assembly model*) [1]. In this model, the separate entities are *tiles*, adorned with *glues*. The intuition is that tiles wander about randomly, and when tiles with matching glues meet, the tiles bind to form a larger assembly; further, such larger structures wander about and may bind to other larger assemblies or tiles.

The main measure of complexity is the number of unique types of tiles necessary and sufficient to uniquely assemble the shape, termed the *tile complexity*. The *temperature* of a system, denoted by τ, is the minimum required binding strength between two entities to enforce a stable attachment; the sum of the strengths between the shared glues of two assemblies must meet or exceed the temperature. Some studies of the model include *negative-strength glues* [2, 8, 12–15, 20], which are repulsive forces which act against a particular bond between two assemblies. Studies of these repulsive forces are motivated by experimental implementation of self-assembly systems which exhibit this behavior [16].

Our contributions. We give one tile set with a constant number of distinct tile types which satisfies the following: given any finite connected shape $S \subset \mathbb{Z}^2$, there exists an assignment of strengths between glues (a glue function) and a temperature τ such that the system uniquely assembles S. The system encodes the shape in its temperature parameter τ and its glue function. Then, by utilizing the inclusion of one negative-strength glue type, the system assembles a width-τ assembly. This width-τ assembly is utilized as a seed for a tile set designed by [23] which "runs" the program encoded by the seed to assemble the shape. This work is the first to show that any shape can be built with a constant number of distinct tile types (where the glues are a function of τ) at any scale without a staged model[5], i.e., it is the first to achieve this in a fully "hands-off" model which requires no experimenter intervention during the assembly process.

Previous results. For self-assembling a shape S, we list the previously known results, which do not use negative glues and use $\mathcal{O}(1)$ temperature unless otherwise specified. Let $K(S)$ be the Kolmogorov complexity[6] of S, and let $T(S)$ be the (smallest) runtime of a Kolmogorov-optimal program outputting S. A tile complexity of $\Theta\left(\frac{K(S)}{\log K(S)}\right)$ is known, using a scale factor of $T(S)$ [23]. With negative-strength glues, a tile complexity of $\Theta\left(\frac{K(S)}{\log K(S)}\right)$ is known,

[5] In the staged self-assembly model, the results of [3] give a construction which can effectively use $\mathcal{O}(1)$ tile types to assemble the shape by increasing the number of bins and stages used.

[6] The Kolmogorov complexity of S is the number of bits in the smallest program which outputs S w.r.t. a universal Turing machine. For more information on Kolmogorov complexity, see [11].

using a $\mathcal{O}(1)$ scale factor [12]. In a *staged* version of the model, where several self-assembly systems are run in parallel across a series of *bins* and then mixed together in *stages*, a tile complexity of $\Theta\left(\frac{K(S)}{\log K(S)}\right)$ at scale factor $T(S)$ is known for $\mathcal{O}(1)$ bins and $\mathcal{O}(1)$ stages along with a method for (optimally) reducing the number of sufficient and necessary tile types by increasing the number of bins and stages [3].

In another staged version of the model, where tiles are partitioned into `DNA` and `RNA` types, and `RNA` types may be "washed away" at a given stage, a tile complexity of $\Theta\left(\frac{K(S)}{\log K(S)}\right)$ at scale factor $\mathcal{O}(\log |S|)$ is known [7]. In a staged model where the self-assembly process is controlled by a chemical reaction network which activates and deactivates tiles' binding sites, a tile plus reaction network complexity of $\Theta\left(\frac{K(S)}{\log K(S)}\right)$ at scale factor $\mathcal{O}(1)$ is known [19].

Related work in high-temperature[7] self-assembly. The first studies of utilizing temperature to encode information involved the "online", mid-assembly-process changing of temperatures [10, 24]. Our result utilizes a high temperature bonding threshold for self-assembly attachment, which we leverage to encode precise information for guiding the self-assembly process through precisely set glue strengths. A number of recent related works have also studied the effects of higher temperature self-assembly systems within various models. Within the aTAM, larger temperatures have been shown to affect the possible behavior of systems [4], and the tile complexity of self-assembled shapes [22]. Within the 2HAM, unique-assembly verification has been shown to be hard for high-temperature systems [21], while the dynamics of certain higher-temperature systems have been shown to be impossible to simulate at lower temperatures [6].

2 Definitions and Model

In this section we first define the two-handed tile self-assembly model with both negative and positive strength glue types. We also formulate the problem of designing a tile assembly system that constructs a constant-scaled shape given the optimal description of that shape.

Tiles and Assemblies. A tile is an axis-aligned unit square centered at a point in $\mathbb{Z}^2$, where each edge is labeled by a *glue* selected from a glue set Π. A *strength function* $\mathrm{str} : \Pi \to \mathbb{Z}$ denotes the *strength* of each glue. Two tiles equal up to translation have the same *type*. A *positioned shape* is any subset of $\mathbb{Z}^2$. A *positioned assembly* is a set of tiles at unique coordinates in $\mathbb{Z}^2$, and the positioned shape of a positioned assembly A is the set of those coordinates. For a given positioned assembly Υ, define the *bond graph* G_Υ to be the weighted grid graph in which each element of Υ is a vertex and the weight of an edge between tiles is the strength of the matching coincident glues or 0.[8] A positioned assembly C is τ*-stable* for positive integer τ provided the bond graph G_C has min-cut at least τ.

For a positioned assembly A and integer vector $\vec{v} = (v_1, v_2)$, let $A_{\vec{v}}$ denote the positioned assembly obtained by translating each tile in A by vector $\vec{v}$. An *assembly* is a translation-free

[7] We say high-temperature self-assembly for consistency with previous literature. The term high temperature may be misleading; e.g., we are not attempting to model what happens in DNA-based self-assembly systems when the literal temperature of the system is raised to high values. Intuitively, higher temperature in this model implies more fine-grained glue strengths. Another natural way to think of high temperature is to fix the temperature to one, but allow rational glue strengths.

[8] Note that only matching glues of the same type contribute a non-zero weight, whereas non-equal glues always contribute zero weight to the bond graph. Relaxing this restriction has been considered [5].

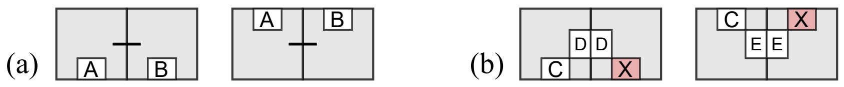


Figure 1 The black lines between two tiles indicate unique unimportant τ-strength bonds. If $\tau = 2$, $str(A) = 1$ and $str(B) = 1$, then the two assemblies in (a) are τ-combinable, since $str(A) + str(B) \geq \tau$ and the positioned assemblies may be translated such that the A and B glues are aligned – such a combination is termed *cooperative binding*, since neither the A nor B glue are alone sufficient to satisfy τ-combination. In (b), we consider two cases concerning the negative strength glue X. If $\tau = 2$, $str(C) = 2$, and $str(X) = -1$, then the assemblies in (b) are not τ-combinable since $str(C) + str(X) < \tau$. If $\tau = 1$, $str(C) = 2$, $str(D) = 2$, $str(E) = 1$ and $str(X) = -1$, then the assemblies are τ-combinable since $str(C) + str(X) \geq \tau$; however, the resultant assembly is unstable, since a cut along the X and E glue has strength $str(X) + str(E) < \tau$; this violates the valid growth-only system definition.

version of a positioned assembly, formally defined to be a set of all translations $A_{\vec{v}}$ of a positioned assembly A. An assembly is τ-stable if and only if its positioned elements are τ-stable. A *shape* is the set of all integer translations for some subset of $\mathbb{Z}^2$, and the shape of an assembly A is defined to be the set of the positioned shapes of all positioned assemblies in A. The *size* of either an assembly or shape X, denoted as $|X|$, refers to the number of elements of any positioned element of X.

Combinable Assemblies. Two assemblies are τ*-combinable* provided they may attach along a border whose strength sums to at least τ. Formally, two assemblies A and B are τ-*combinable* into an assembly C provided $G_{C'}$ for any $C' \in C$ has a cut (A', B') of strength at least τ for some $A' \in A$ and $B' \in B$. We call C a *combination* of A and B. Figure 1 gives examples of combinable and not combinable assemblies.

Two Handed Assembly Model: Growth-only Version

A *two-handed tile assembly system (2HAM system)* is an ordered pair (T, τ) where T is a set of single tile assemblies, called the *tile set*, and $\tau \in \mathbb{N}$ is the *temperature*. In the *growth-only* model, assembly proceeds by repeated combination of assembly pairs to form new assemblies starting from the initial tile set. The *producible assemblies* are those constructed in this way.

▶ **Definition 1** (2HAM Producibility (growth-only)). For a given 2HAM system $\Gamma = (T, \tau)$, the set of *producible assemblies* of Γ, denoted $\texttt{PROD}_\Gamma$, is defined recursively:
- (Base) $T \subseteq \texttt{PROD}_\Gamma$
- (Combinations) For any $A, B \in \texttt{PROD}_\Gamma$ such that A and B are τ-combinable into C, then $C \in \texttt{PROD}_\Gamma$.

The inclusion of negative glues, in general, allows for unstable assemblies to be producible. In previous literature, such assemblies "detach", forming two new producible assemblies. We impose the following constraint on growth-only systems which disallows production of unstable assemblies which would fall apart. Satisfying the growth-only constraint argues that the system has simpler kinetics than a non-growth-only system since the system does not rely on detachment events.

For a system $\Gamma = (T, \tau)$, we say $A \rightarrow_1^\Gamma B$ for assemblies A and B if A is τ-combinable with some producible assembly to yield B, or if $A = B$. Intuitively this means that A may grow into assembly B through one or fewer combination. We define the relation $\rightarrow^\Gamma$ to be the transitive closure of $\rightarrow_1^\Gamma$, ie., $A \rightarrow^\Gamma B$ means that A may grow into B through a sequence of combinations.

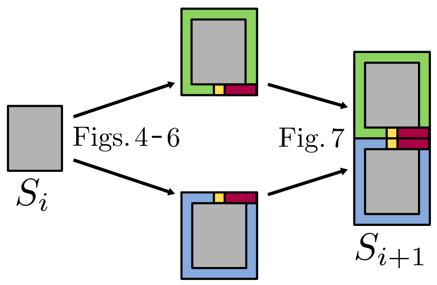


Figure 2 A simplified overview of the growing step. S_i is a width-$\Theta(i)$, height-$\Theta(2^i)$ assembly with particular exposed edge glues. S_i nondeterministically assembles one of two assemblies; a *top* and *bottom*. The top and bottom share one glue of strength $2\tau - 1$ shown in yellow, and i many -1 strength glues shown in red. Thus, the top and bottom bind with strength $2\tau - i - 1$, which is τ-stable only if $i < \tau$. The resultant assembly adds two width and doubles the height of S_i, so its dimensions are $\Theta(i + 1) \times \Theta(2^{i+1})$. Further, its exposed glues allow the process to repeat.

▶ **Definition 2** (Valid Growth-Only System). A 2HAM system $\Gamma = (T, \tau)$ is a *valid growth-only* system if for all $A \in \texttt{PROD}_\Gamma$, A is τ-stable.

▶ **Definition 3** (Terminal Assemblies). A *terminal* assembly of a valid growth-only 2HAM system is a producible assembly that cannot combine with any other producible assembly. Formally, an assembly $A \in \texttt{PROD}_\Gamma$ of a 2HAM system $\Gamma = (T, \tau)$ is *terminal* provided A is not τ-combinable with any producible assembly of Γ.

We formalize what it means for a 2HAM system to uniquely build a given assembly or a given shape.

▶ **Definition 4** (Unique Assembly). A 2HAM system *uniquely* produces an assembly A if all producible assemblies have a growth path towards the terminal assembly A. Formally, a 2HAM system $\Gamma = (T, \tau)$ *uniquely* produces an assembly A provided that A is terminal, and for all $B \in \texttt{PROD}_\Gamma$, $B \rightarrow^\Gamma A$.

▶ **Definition 5** (Unique Shape Assembly[9]). A 2HAM system uniquely produces a shape S if all producible assemblies have a growth path to a terminal assembly of shape S. Formally, a 2HAM system $\Gamma = (T, \tau)$ *uniquely assembles* a finite shape S if for every $A \in \texttt{PROD}_\Gamma$, there exists a terminal $A' \in \texttt{PROD}_\Gamma$ of shape S such that $A \rightarrow^\Gamma A'$.

3 Assembly of General Shapes with Constant Tiles

Here we give the main construction of the paper. First presented is our key contribution – assembly of a precise-width rectangle – detailed in Subsection 3.1, followed by its composition with established techniques from [23] for the main result in Subsection 3.2.

3.1 Key idea: precise-width rectangle using $\mathcal{O}(1)$ tile types

Here, we present a construction for building a precise-width rectangle from a constant-bounded set of tile types. Note that the convention in this paper is width $\times$ height. Formally,

[9] Some previous literature calls this *strict self-assembly*, typically to contrast another definition, *weak self-assembly*; we choose the name unique shape assembly to contrast unique assembly.

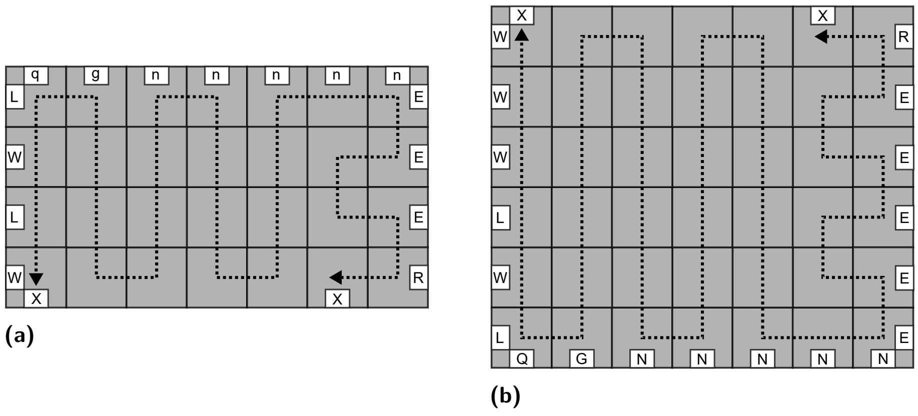


Figure 3 The base assembly, shown in two separate subassemblies; (a) shows the top subassembly, and (b) the bottom. The two subassemblies combine using cooperative binding at the strength-$\lceil \frac{\tau}{2} \rceil$ glues labeled X. The dotted line indicates distinct tile types which attach along the path with full τ strength glues. The snaking pattern ensures that each subassembly is complete before both X glues are available. Once these two subassemblies bind, the resultant assembly satisfies S_0.

▶ **Lemma 6.** *Given a temperature $\tau > 2$, there exists a negative glue, growth-only $2HAM$ tile system $\Gamma = \{T, \tau\}$ such that $|T| = \mathcal{O}(1)$ and Γ uniquely produces an assembly which is a* $(18 + 4\tau) \times (2^{\tau+5} - 6)$ *rectangle.*

Proof. We give a proof by construction. Unless explicitly stated otherwise, all glues have strength $\lceil \frac{\tau}{2} \rceil$, so at least two matching glues are required for a τ-stable attachment. This is called *cooperative binding.* The construction is split into two steps: *growing* and *finishing.* Figure 2 shows a simplified overview of the growing step. The growing step of the construction concerns producing an assembly with width $7+2\tau$, through a process consisting of τ iterations of growth, each adding a constant-bounded width to the assembly. The $i+1^{\text{th}}$ iteration of growth is initiated by a combination of two assemblies with total binding strength $2\tau - i - 1$; thus, after the τ^{th} iteration of growth, the binding strength which would initiate the next iteration of growth has total binding strength $2\tau - \tau - 1 < \tau$, and growth halts.

The finishing step involves the system's "detecting" that the growth process has completed. This is achieved using the following technique: by adding a total strength of 1 shared between two assemblies at each iteration of growth, once the assemblies have completed τ repetitions of growth, they bind with strength τ. This step also gives the system its property of unique assembly of an assembly whose shape is a rectangle (and not just unique shape assembly). That is, there is exactly one terminal assembly of the system – as opposed to several terminal assemblies with the correct rectangular shape. Maintaining this property in this lemma is required to achieve the same property in Theorem 7.

3.1.1 Growing

The construction is described and proven correct via induction. The induction is on iterations of rectangular assemblies with well-defined exposed glues, termed S_i. Formally, S_i refers to a $(7 + 2i) \times (2^{i+4} - 6)$ rectangular assembly with the following exposed glue labels, written as strings built by concatenating the glue labels in left-to-right/top-to-bottom order):

- North glues: $qp^i gn^{i+5}$
- East glues: $E^{2^{i+3}-5}R^2E^{2^{i+3}-3}$
- South glues: QP^iGN^{i+5}
- West glues: $LW^{2^{i+3}-7}LW^4LW^{2^{i+3}-7}L$

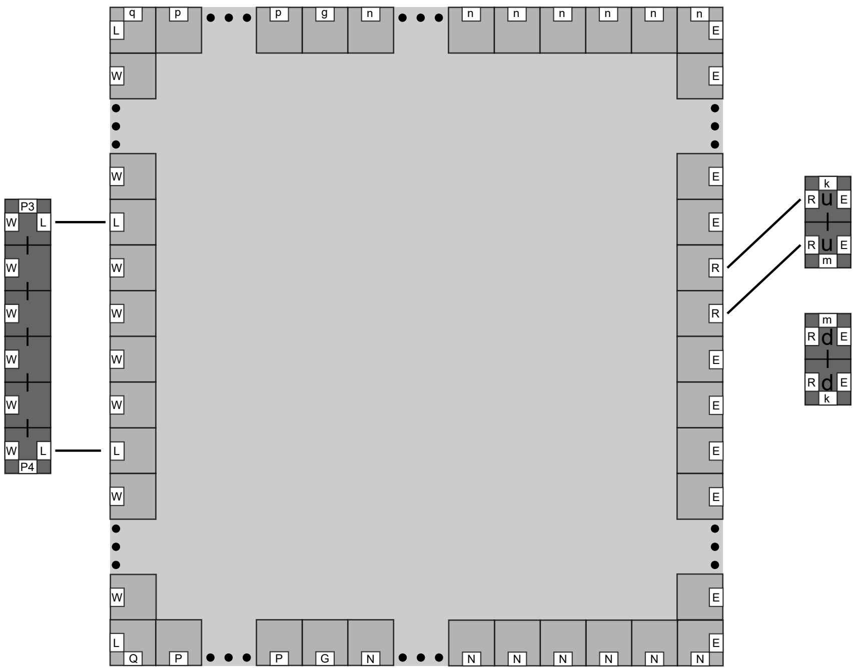


Figure 4 An assembly satisfying S_i. The dots indicate an omitted set of repeated tiles; e.g., the dots between the tiles exposing p indicate the omission of the i glues with label p. On the right are the keystones, which attach cooperatively using R glues. Only one keystone may attach, introducing nondeterminism; this is how the producibility of two assemblies, a top and bottom assembly, are implied by the production of one assembly satisfying S_i.

The goal is to show that if an assembly satisfying S_i is producible, then an assembly satisfying S_{i+1} is producible iff $i < \tau$. Further, only $\mathcal{O}(1)$ tile types are used in the inductive step. Then it suffices to show that S_0 is producible in $\mathcal{O}(1)$ tile types, implying S_τ is producible, which has width $\times$ height as in the lemma statement.

Base case. An assembly satisfying S_0 is shown to be trivially assembled by a set of $\mathcal{O}(1)$ tile types in Figure 3.

Inductive step. The next three paragraphs describe the inductive step. The goal is to show that if S_i is producible, then a top and bottom assembly are producible which can bind to produce S_{i+1} iff $i < \tau$. Consider an assembly with exposed glues satisfying S_i. The two R glues exposed allow attachment of a *keystone* assembly via cooperative binding as seen in Figure 4. There are two keystone types: *up* and *down*. Only one may attach. The L glues in the middle of the west-side exposed glues, spaced by four W glues, also allow the attachment of a supertile using cooperative binding. Once the keystone or west-side supertile has attached, a single tile type suffices to attach tiles along any long set of repeating glues on the assembly (e.g., the E glues on the east side), until the glue is no longer available, as seen in Figure 5. This type of single tile type attaching along arbitrary walls is termed *propagation* of a tile.

The tiles which propagate to the corners of the west side of the assembly allow the attachment of three tiles around the corner which allow propagation of tiles along the north and south faces of the assembly. Once these tiles propagate, depending on which keystone was attached to the assembly, the attachment of a *tooth* occurs on the corresponding face (e.g., on

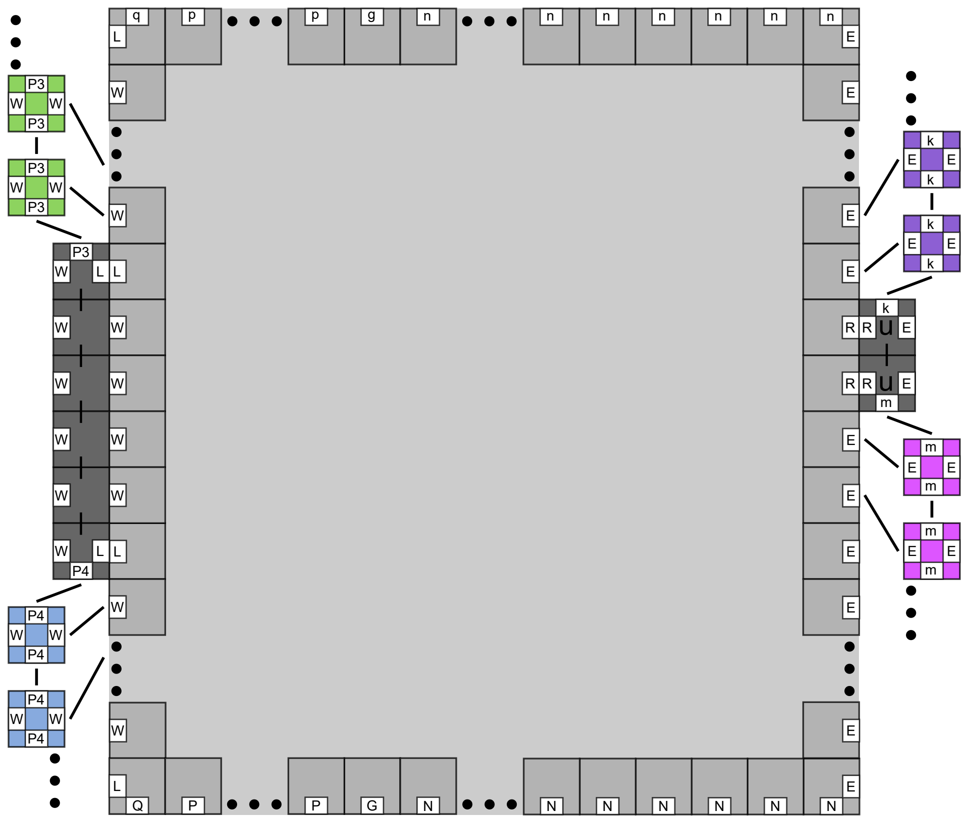


Figure 5 Once the keystone and supertile on the left have attached, a sequence of single tile attachments may occur using cooperative binding at newly available glues. The tiles are designed such that they may attach along arbitrarily long faces, as long as the appropriate glue is exposed (e.g., a W glue in the top-left tile's case).

the north face if the up keystone was attached) as shown in Figure 6. The tooth is a supertile with a specific geometry which will be motivated later on. The tooth initiates a propagation of tiles along the corresponding face. The result is the production of two assemblies, one having attached the up keystone and attached all previously discussed propagating tiles and supertiles, and one having attached the down keystone and similar tiles. We refer to the former as a *top* assembly, and the latter as a *bottom* assembly.

When a top assembly and bottom assembly attach, the result is an assembly satisfying S_{i+1}. A top assembly and bottom assembly are designed to attach iff $i < \tau$. When $i \geq \tau$, the binding strength between a top and bottom assembly is $\tau - 1$, and thus is insufficient. This design can be seen in Figure 7. Note that the -1 glues are propagated via the N-labeled glues from the base assembly. Since S_i has $i + 5$ many N glues exposed, 5 of which are covered by the tooth, the bottom/top assembly which assembles from S_i exposes i many -1 glues. Then the bonding strength between top and bottom assemblies is $2\tau - 1 - i$, which is less than τ iff $i \geq \tau$. The complementary geometry of the teeth ensure that a top and bottom assembly which are assembled from S_i and S_j respectively, with $i \neq j$, cannot align their a glues and will not attach.

Dimensions of S_i. The base assembly satisfying S_0 is 7×10. When a top and bottom assembly attach, both have added 2 width in tiles; one tile width propagated on the west side, and one on the east. Then the width of S_i is $7 + 2i$. A top and bottom assembly which have

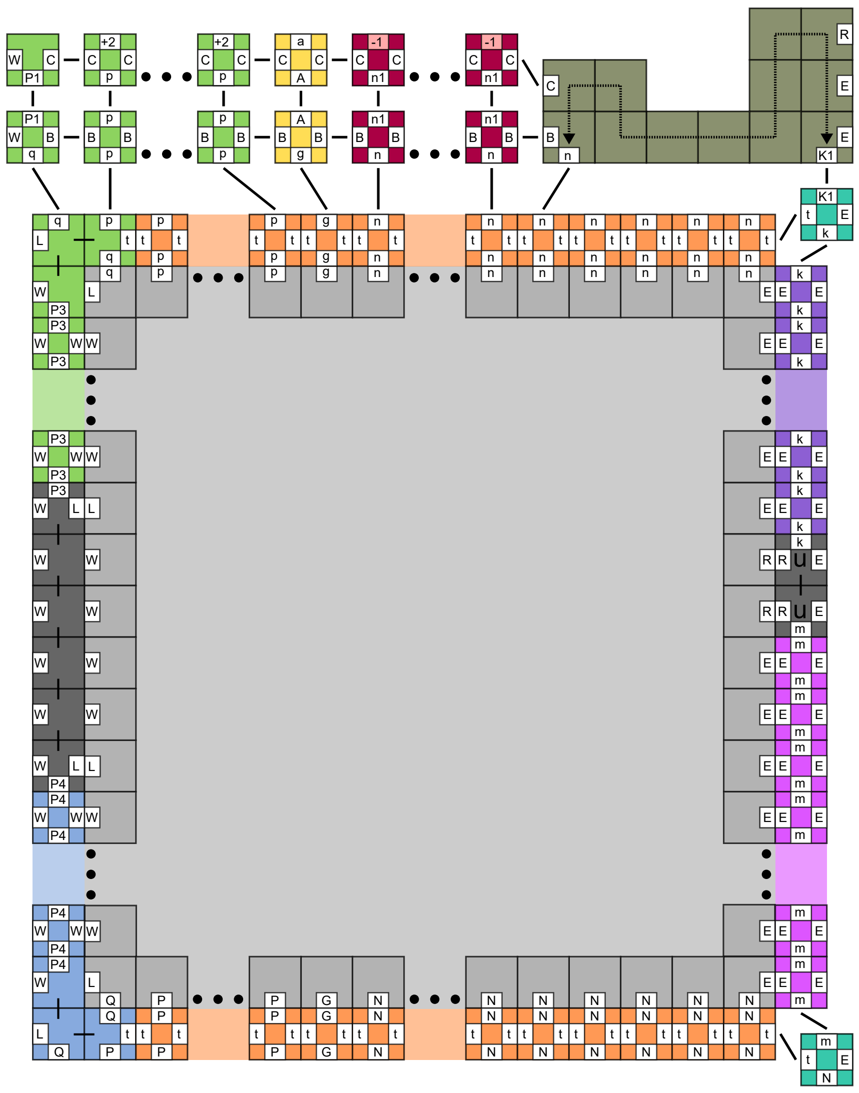


Figure 6 At the top-right and bottom-right corners, tiles attach which indicate that the left-hand side tile propagation has reached the right-hand side of the assembly. In the case of an assembly which has attached an up keystone, the tooth attaches on the top side of the assembly, and initiates a propagation of tiles along the top face. A tooth with complementary geometry will attach on the bottom side of the assembly if a down keystone attaches instead, as can be seen in Figure 7.

combined add 6 height in tiles on top of doubling in height: 2 via tile propagation on the top and bottom, and 4 along where the top and bottom assemblies attach. Then the height of S_i, $h(i)$, is defined by the recurrence $h(i) = 2h(i-1) + 6$ with $h(0) = 10$. Solving the recurrence gives a height of $2^{i+4} - 6$. Then consider a combination of a top and bottom assembly which formed from some assembly satisfying S_{i-1}; the resultant dimension is $(7 + 2i) \times (2^{i+4} - 6)$.

3.1.2 Finishing

When a top and bottom assembly combine to form an assembly satisfying S_τ, the growing step shows that the process will not continue to produce $S_{\tau+1}$. However, the attachment of a supertile on the west side, a keystone, and the resultant tile propagation still occurs.

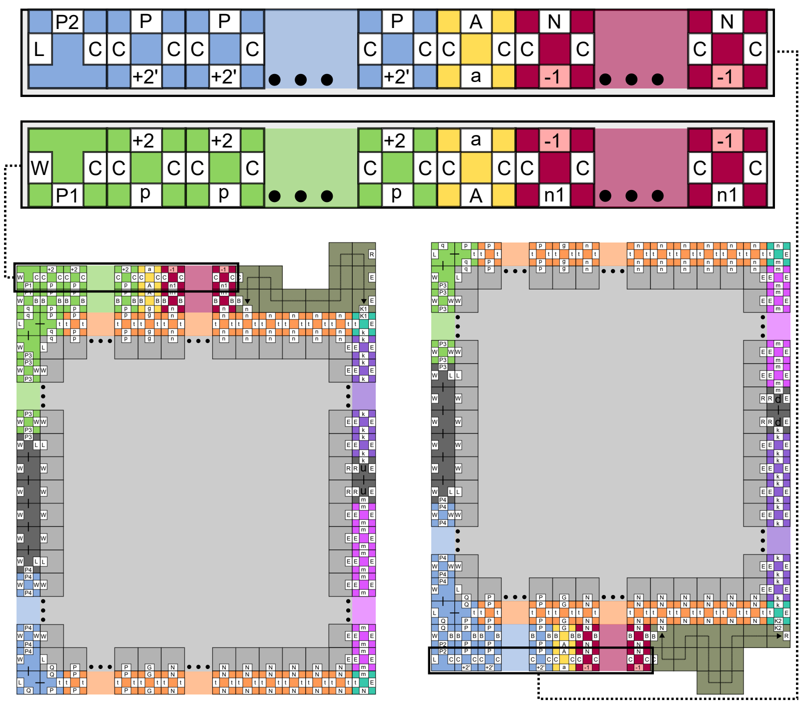


Figure 7 A bottom assembly (left) and top assembly (right). At the top of the figure are the tiles whose glues bond a bottom and top assembly; in particular, the a and -1 glues, with $str(a) = 2\tau - 1$ and $str(-1) = -1$. A top and bottom assembly grown from an assembly S_i each expose i glues with strength -1. Then the strength of the attachment between them is $2\tau - i - 1$, and is sufficient when $i < \tau$ but insufficient when $i = \tau$. Note that $+2$ and $+2'$ glues do not match; their purpose is described later in the finishing step.

The teeth attach and so do the tiles which propagate resulting from the attachment of a tooth. Then an assembly satisfying S_τ implies the production of the corresponding top and bottom assemblies. These assemblies are not rectangular. This step involves detecting that the iterative process has reached τ repetitions, and the system should finish its rectangle.

Figure 8 gives an overview of the finishing step. The technique discussed in the growing phase is employed by two disjoint tile sets, one called *system 1* and the other *system 2*. The sets of glues on the tiles in the two systems are disjoint except for two glues: the -1 glue, and the $+2$ glue described but not used in the growing phase. In the growing phase, recall that on the north face of a bottom assembly of system 1 which assembled from S_i, there are i many strength 2 glues labeled $+2$ exposed which are not used (recall Figure 7). These glues are designed to match with the corresponding glues in system 2. Then the strength of binding between these shared glues is $2i - i = i$. Thus, only when $i \geq \tau$ is this binding τ-stable. Similarly, system 1's top assembly attaches with the system 2's bottom assembly under the same constraint. These resultant assemblies expose cooperative binding locations which were not present before this attachment, allowing these two new assemblies to combine, and then fill into a rectangle using a $\mathcal{O}(1)$-sized tile set. Next, we give the dimensions of the completed rectangular assembly: system 1's top and system 2's bottom assembly, once attached, form a $(7 + 2(\tau + 1)) = (9 + 2\tau) \times (2^{(\tau+1)+4} - 6) = (2^{\tau+5} - 6)$ assembly – this can be derived from the combination of two assemblies satisfying S_τ assembling into an assembly satisfying $S_{\tau+1}$ not in exposed glues, but in size. System 1's top and system 2's bottom are combined with system 1's bottom and system 2's top into one via a width-two column, resulting in a $2(9 + 2\tau) + 2 = 18 + 4\tau \times 2^{\tau+5} - 6$ assembly. ◀

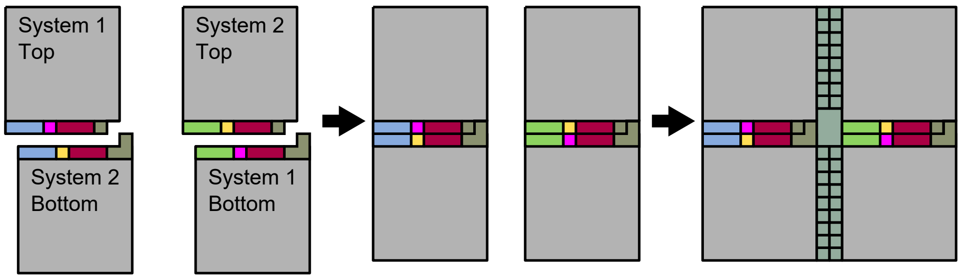


Figure 8 An overview of the finishing step. The system described in the growing step, denoted here as system 1, is repeated, denoted system 2, such that the only matching glues between the two systems are a strength-2 glue type and the one strength-(-1) glue type. Between a system 1 top/bottom and system 2 bottom/top assembled from S_i, the number of shared strength-2 glues and strength-(-1) glues is i, so the sum of strengths between shared glues is $2i - i = i$. This allows the top/bottom assembly of system 1 to make a τ-stable attachment to the bottom/top of system 2 only after each system assembles S_τ, thus detecting when the rectangle has $\Theta(\tau)$ width. Once these tops and bottoms attach, new cooperative binding locations initiate a constant-sized set of tiles to bind the two rectangles, simply to satisfy unique assembly of the rectangle.

3.2 From rectangle to shape

To assemble the target shape, a technique is combined with Lemma 6. The technique, shown by Soloveichik and Winfree [23], first assembles a *seed block* bearing a representation (a series of exposed glues) of a Kolmogorov-optimal program which computes the spanning tree of S. A $\mathcal{O}(1)$-sized tile set is used to "run" the program (via Turing machine (TM) simulation) from the seed block. The seed block assembles into a $c \times c$ assembly, logically representing one coordinate of S. Assembly proceeds from the seed block in a subset of the four cardinal directions depending on the spanning tree computed by the program. If the spanning tree has an edge in a direction to a coordinate adjacent to the seed block, a $c \times c$ *growth block* is assembled in that direction. Each time a growth block is assembled, the program is run again to determine which adjacent coordinates (w.r.t. the growth block) are connected by edges in the spanning tree; if so, a growth block is assembled in that direction. After assembling all growth blocks, the unique assembly is a $c \times c$ scaled version of S.

The seed block of [23] is assembled using $\mathcal{O}(\frac{K(S)}{\log K(S)})$ tile types. In our case, the seed block is assembled using the $\mathcal{O}(1)$-sized tile set of Lemma 6: we assemble a rectangle of width n where n is the length of a unary encoding of the Kolmogorov-optimal program. This seed is combined with the $\mathcal{O}(1)$-sized tile set which runs the program and assembles the shape. Figure 9 is a simplified overview of constructing a seed block compatible with a TM simulating tileset. The formal result is as follows:

▶ **Theorem 7.** *Given a shape S, there exists a negative glue, growth-only 2HAM tile system $\Gamma = \{T, \tau\}$ with $|T| = \mathcal{O}(1)$ whose unique assembly has shape S at some scale factor.*

Proof. The system is a union of the Lemma 6 system and a subset of the TM simulating tile set of [23]. If the entire TM simulating tile set is added to the system, depending on the seed block, some tiles may never bind to the seed block, and thus do not grow into the target shape and violate unique assembly definition. We include the subset of the TM simulating tile set which will be used by the program encoded by the seed block. Observe that the unique assembly produced by the system of Lemma 6 is not a square, nor does it have any exposed glues designed to bind with the TM simulation tile set. In order to assemble a square from the terminal assembly of Lemma 6, two such rectangular assemblies

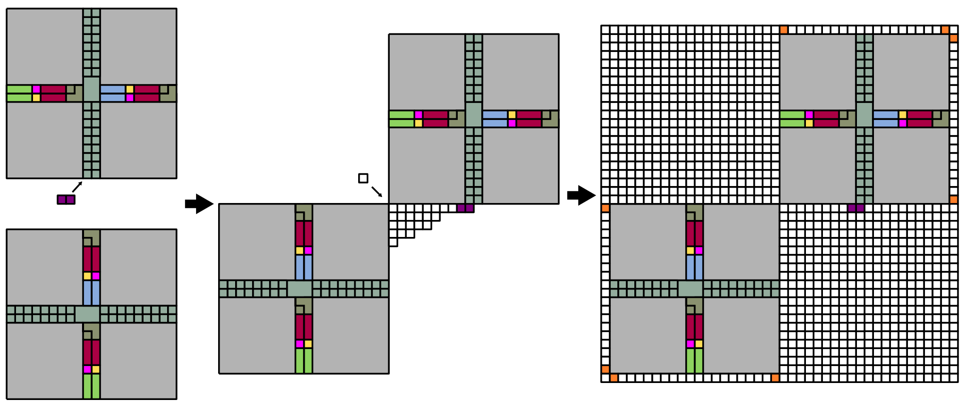


Figure 9 The combination of two Lemma 6 constructions into a seed block. The two-tile assembly in the first subfigure initializes the attachment of the set of white tiles, which indicate a constant-sized set of *filler* tiles which are used to fill in a full square. Once the square is filled in, new cooperative binding locations are exposed where the filler tiles meet the non-filler tiles. At this location, tiles begin to propagate, adding a one-tile perimeter to the assembly. The orange tiles on the outmost perimeter of the rightmost figure demarcate the beginning and ending of glues exposing the unary program which constructs the shape S via the TM simulation of [23]. The rest of the perimeter exposes glues which the TM simulation ignores.

are assembled in parallel, one which is rotated 90 degrees – this "rotation" is w.r.t. the other rectangular assembly and the way the two assemblies will bind. These two assemblies combine and then "fill-in" to a square trivially using a constant-sized tile set – similar to propagation of tiles along an edge, cooperative binding can be used to add tiles between two assembled rectangles to assemble a square (technique first used in [18]). Once assembly of the square is complete, more tile propagation via another constant-sized tile set assembles a one-tile perimeter which exposes glues – described in the following paragraph – which allow the assembly to act as a seed block similar to the Soloveichik and Winfree construction [23].

Let P be the (binary) program used to assemble S via the construction of [23], R be some mapping from binary strings to unary strings, and R' be some mapping from unary strings $\{1\}^i$ with $i \in \mathbb{N}$ to unary strings $\{1\}^j$ with $j = 20 + 4m$ for $m \in \mathbb{N}$ – the intuition for the mapping R' is to map arbitrary unary strings to numbers which are widths of assemblies assembled by Lemma 6.

Once the square seed block is assembled and a one-tile perimeter is attached, three glue types are exposed: 1, b, and λ. Across the length of filler tiles (those in the square which are not from the Lemma 6 construction), λ glues are placed; these symbols are ignored by the TM simulating tile set. The b glues are placed at the beginnings and ends of the edges of the square; these indicate where to start the TM simulation. Along the edges of the rectangles from the Lemma 6 construction, glues labeled 1 are placed; these 1 glues are the relevant glues logically. The TM simulation converts these to unary strings by R'^{-1}, and then to P by R^{-1}. Then the TM simulation tiles run the program P which assembles the shape. ◀

4 Future Work

The most apparent direction for future work is to achieve the unique assembly of shapes at a $\mathcal{O}(1)$ scale factor with $\mathcal{O}(1)$ tile types. This result may be achieved through a combination of the techniques used in this work and in [12], which achieves $\mathcal{O}(1)$-scale factor with $\mathcal{O}(\frac{K(S)}{\log K(S)})$

tile types where $K(S)$ is the Kolmogorov complexity of the shape S. Their construction utilizes a dynamic behavior of negative glues not utilized in this work called "breaking": the combination of two assemblies may result in an unstable assembly, which then breaks into two assemblies – a formal model and some usages of breakage may be seen in [2, 8, 20]. Their usage of breaking involves performing a computation – via a self-assembly process which simulates a TM – which builds the shape pixel-by-pixel (using $\mathcal{O}(1)$-sized assemblies per pixel), and then breaks the TM simulating assembly into $\mathcal{O}(1)$-sized pieces, leaving the shape S at a $\mathcal{O}(1)$ scale factor (along with "small garbage" of $\mathcal{O}(1)$ size). That technique may be applicable to the construction given in this work in order to break the precise-width rectangles after they are used as input for the TM which outputs S.

Another direction might be to achieve the unique assembly of scaled shapes with $\mathcal{O}(1)$ tile types using only positive-strength glues. We have briefly discussed previous positive-strength results which use $\Theta\left(\frac{K(S)}{\log K(S)}\right)$ tile types. Could this be lowered to $\mathcal{O}(1)$ tile types by calibrating the temperature and glue strengths, or is there some super-constant lower bound that cannot be breached?

References

1 Sarah Cannon, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Matthew J. Patitz, Robert Schweller, Scott M. Summers, and Andrew Winslow. Two hands are better than one (up to constant factors): Self-assembly in the 2ham vs. atam. In Natacha Portier and Thomas Wilke, editors, *STACS*, volume 20 of *LIPIcs*, pages 172–184. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.STACS.2013.172.

2 Cameron Chalk, Erik D. Demiane, Martin L. Demaine, Eric Martinez, Robert Schweller, Luis Vega, and Tim Wylie. Universal shape replicators via self-assembly with attractive and repulsive forces. In *Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'17)*, 2017.

3 Cameron Chalk, Eric Martinez, Robert Schweller, Luis Vega, Andrew Winslow, and Tim Wylie. Optimal staged self-assembly of general shapes. *Algorithmica*, May 2017. doi:10.1007/s00453-017-0318-0.

4 H.-L. Chen, D. Doty, and S. Seki. Program size and temperature in self-assembly. *Algorithmica*, 72(3):884–899, 2015.

5 Qi Cheng, Gagan Aggarwal, Michael H. Goldwasser, Ming-Yang Kao, Robert T. Schweller, and Pablo Moisset de Espanés. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34:1493–1515, 2005.

6 Erik D. Demaine, Matthew J. Patitz, Trent A. Rogers, Robert T. Schweller, Scott M. Summers, and Damien Woods. The two-handed tile assembly model is not intrinsically universal. *Algorithmica*, 74(2):812–850, 2016.

7 Erik D. Demaine, Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers. Self-assembly of arbitrary shapes using rnase enzymes: Meeting the kolmogorov bound with small scale factor (extended abstract). In *Proc. of the 28th International Symposium on Theoretical Aspects of Computer Science (STACS'11)*, 2011.

8 David Doty, Lila Kari, and Benoît Masson. Negative interactions in irreversible self-assembly. *Algorithmica*, 66(1):153–172, 2013. doi:10.1007/s00453-012-9631-9.

9 Constantine Evans. *Crystals that Count! Physical Principles and Experimental Investigations of DNA Tile Self-Assembly*. PhD thesis, California Inst. of Tech., 2014.

10 Ming-Yang Kao and Robert T. Schweller. Reducing tile complexity for self-assembly through temperature programming. In *SODA 2006: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 571–580, 2006.

11 M. Li and P. Vitanyi. *An Introduction to Kolmogorov and Its Applications (Second Edition)*. Springer Verlag, New York, 1997.

12 Austin Luchsinger, Robert Schweller, and Tim Wylie. Self-assembly of shapes at constant scale using repulsive forces. In Matthew J. Patitz and Mike Stannett, editors, *Unconventional Computation and Natural Computation*, pages 82–97, Cham, 2017. Springer International Publishing.

13 Matthew J. Patitz, Trent A. Rogers, Robert Schweller, Scott M. Summers, and Andrew Winslow. Resiliency to multiple nucleation in temperature-1 self-assembly. In *DNA Computing and Molecular Programming*. Springer International Publishing, 2016.

14 Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers. Exact shapes and turing universality at temperature 1 with a single negative glue. In *DNA Computing and Molecular Programming*, volume 6937 of *LNCS*, pages 175–189. Springer, 2011.

15 John H. Reif, Sudheer Sahu, and Peng Yin. Complexity of graph self-assembly in accretive systems and self-destructible systems. *Theoretical Comp. Sci.*, 412(17):1592–1605, 2011. doi:10.1016/j.tcs.2010.10.034.

16 Paul W. K. Rothemund. Using lateral capillary forces to compute by self-assembly. *Proceedings of the National Academy of Sciences*, 97(3):984–989, 2000. doi:10.1073/pnas.97.3.984.

17 Paul W. K. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, March 2006. doi:10.1038/nature04586.

18 Paul W. K. Rothemund and Erik Winfree. The program-size complexity of self-assembled squares (extended abstract). In *Proc. of the 32nd ACM Sym. on Theory of Computing*, STOC'00, pages 459–468, 2000.

19 Nicholas Schiefer and Erik Winfree. *Universal Computation and Optimal Construction in the Chemical Reaction Network-Controlled Tile Assembly Model*, pages 34–54. Springer International Publishing, Cham, 2015. doi:10.1007/978-3-319-21999-8_3.

20 Robert Schweller and Michael Sherman. Fuel efficient computation in passive self-assembly. In *SODA 2013: Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1513–1525. SIAM, 2013.

21 Robert Schweller, Andrew Winslow, and Tim Wylie. Complexities for high-temperature two-handed tile self-assembly. In Robert Brijder and Lulu Qian, editors, *DNA Computing and Molecular Programming*, pages 98–109, Cham, 2017. Springer International Publishing.

22 Sinnosuke Seki and Yasushi Ukuno. On the behavior of tile assembly system at high temperatures. *Computability*, 2(2):107–124, 2013.

23 David Soloveichik and Erik Winfree. Complexity of self-assembled shapes. *SIAM Journal on Computing*, 36(6):1544–1569, 2007. doi:10.1137/S0097539704446712.

24 Scott M. Summers. Reducing tile complexity for the self-assembly of scaled shapes through temperature programming. *Algorithmica*, 63(1):117–136, 2012.

A Unified PTAS for Prize Collecting TSP and Steiner Tree Problem in Doubling Metrics

T-H. Hubert Chan[1]
Department of Computer Science, The University of Hong Kong, Hong Kong, China
hubert@cs.hku.hk

Haotian Jiang[2]
Department of Physics, Tsinghua University, Beijing, China
jht14@mails.tsinghua.edu.cn

Shaofeng H.-C. Jiang
The Weizmann Institute of Science, Rehovot, Israel
shaofeng.jiang@weizmann.ac.il
https://orcid.org/0000-0001-7972-827X

Abstract

We present a unified (randomized) polynomial-time approximation scheme (PTAS) for the prize collecting traveling salesman problem (PCTSP) and the prize collecting Steiner tree problem (PCSTP) in doubling metrics. Given a metric space and a penalty function on a subset of points known as terminals, a solution is a subgraph on points in the metric space, whose cost is the weight of its edges plus the penalty due to terminals not covered by the subgraph. Under our unified framework, the solution subgraph needs to be Eulerian for PCTSP, while it needs to be a tree for PCSTP. Before our work, even a QPTAS for the problems in doubling metrics is not known.

Our unified PTAS is based on the previous dynamic programming frameworks proposed in [Talwar STOC 2004] and [Bartal, Gottlieb, Krauthgamer STOC 2012]. However, since it is unknown which part of the optimal cost is due to edge lengths and which part is due to penalties of uncovered terminals, we need to develop new techniques to apply previous divide-and-conquer strategies and sparse instance decompositions.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Doubling Dimension, Traveling Salesman Problem, Polynomial Time Approximation Scheme, Steiner Tree Problem, Prize Collecting

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.15

Related Version A full version of the paper can be found at [11], `https://arxiv.org/abs/1710.07774`.

[1] This work was partially supported by the Hong Kong RGC under the grant 17217716.
[2] This research is supported in part by the National Basic Research Program of China Grant 2015CB358700, the National Natural Science Foundation of China Grant 61772297, 61632016, 61761146003, and a grant from Microsoft Research Asia.

© T-H. Hubert Chan, Haotian Jiang, and Shaofeng H.-C. Jiang;
licensed under Creative Commons License CC-BY

1 Introduction

We study prize collecting versions of two important optimization problems: the prize collecting traveling salesman problem ($\mathsf{PC}^{\mathsf{TSP}}$) and the prize collecting Steiner tree problem ($\mathsf{PC}^{\mathsf{STP}}$). In both problems, we are given a metric space and a set of points called *terminals*, and a non-negative penalty function on the terminals. A solution for either problem is a connected subgraph with vertex set from the metric. In addition, it needs to be an Eulerian (multi-)graph[3]for $\mathsf{PC}^{\mathsf{TSP}}$ and a tree for $\mathsf{PC}^{\mathsf{STP}}$. The cost of a solution is the sum of the weights of edges in the solution plus the sum of penalties due to terminals not visited by the solution.

Prize Collecting Problems in General Metrics. The prize collecting setting was first considered by Balas [4], who proposed the prize collecting TSP. However, the version that Balas considered is actually more general, in the sense that each terminal is also associated with a reward, and the goal is to find a tour that minimizes the tour length plus the penalties, and collects at least a certain amount of rewards. The setting that we consider was suggested by Bienstock et al. [8], and they used LP rounding to give a 2.5-approximation algorithm for the $\mathsf{PC}^{\mathsf{TSP}}$ and a 3-approximation for the $\mathsf{PC}^{\mathsf{STP}}$. Later on, a unified primal-dual approach for several network design problems was proposed [17]; this approach improves the approximation ratios for both $\mathsf{PC}^{\mathsf{TSP}}$ and $\mathsf{PC}^{\mathsf{STP}}$ to 2 in general metrics. The 2-approximation had remained the state of the art for more than a decade, until Archer et al. [1] finally broke the 2 barrier for both problems. Subsequently, in a note [16], Goemans combined their argument with other algorithms, and gave a 1.915-approximation for the $\mathsf{PC}^{\mathsf{TSP}}$, which is the state of the art.

Prize Collecting Problems in Bounded Dimensional Euclidean Spaces. $\mathsf{PC}^{\mathsf{TSP}}$ and $\mathsf{PC}^{\mathsf{STP}}$ are APX-hard in general metrics, because even the special cases, the TSP and the Steiner tree problem, are APX-hard. Although the seminal result by Arora [2] showed that both TSP and STP have PTAS's in bounded dimensional Euclidean spaces, the prize collecting setting was not discussed. However, we do believe that their approach may be directly applied to get PTAS's for the prize collecting versions of both problems, with a slight modification to the dynamic programming algorithms. Later, A PTAS for the Steiner Forest Problem (which generalizes the STP) was discovered by Borradaile et al. [9]. Based on this result, Bateni et al. [7] studied the Prize Collecting Steiner Forest Problem, and gave a PTAS for the special case when the penalties are multiplicative, but this does not readily imply a PTAS for the $\mathsf{PC}^{\mathsf{TSP}}$ or the $\mathsf{PC}^{\mathsf{STP}}$.

Prize Collecting Problems in Special Graphs. Planar graphs is an important class of graphs. Both problems are considered in planar graphs, and a PTAS is presented by Bateni et al. [6] for $\mathsf{PC}^{\mathsf{TSP}}$ and $\mathsf{PC}^{\mathsf{STP}}$. Moreover, they noted that both problems are solvable in polynomial time in bounded treewidth graphs, and their PTAS relies on a reduction to the bounded treewidth cases. They also showed that the Prize Collecting Steiner Forest Problem, which is a generalization of the $\mathsf{PC}^{\mathsf{STP}}$, is significantly harder, and it is APX-hard in planar graphs and Euclidean instances. As for the minor forbidden graphs, which generalizes planar graphs, there are PTAS's for various optimization problems, such as TSP by Demaine et al. [14]. However, the PTAS's for prize collecting problems, to the best of our knowledge, are unknown.

[3] An undirected connected multi-graph is Eulerian, if every vertex has even degree.

Generalizing Euclidean Dimension. Going beyond Euclidean spaces, doubling dimension [3, 13, 18] is a popular notion of dimensionality. It captures the bounded local growth of Euclidean spaces, and does not require any specific Euclidean properties such as vector representation or dot product. A metric space has doubling dimension at most k, if every ball can be covered by at most 2^k balls of half the radius. This notion generalizes the Euclidean dimension, in that every subset of $\mathbb{R}^d$ equipped with ℓ_2 has doubling dimension $O(d)$. Although doubling metrics are more general than Euclidean spaces, recent results show that many optimization problems have similar approximation guarantees for both spaces: there exist PTAS's for the TSP [5], a certain version of the TSP with neighborhoods [12], and the Steiner forest problem [10], in doubling metrics.

Our Contributions. In this paper, we extend this line of research, and give a unified PTAS framework for both $\mathsf{PC}^{\mathsf{TSP}}$ and $\mathsf{PC}^{\mathsf{STP}}$. We use PC^{X} when the description applies to either problem. Our main result is Theorem 1.

▶ **Theorem 1.** *For any $0 < \epsilon < 1$, there exists an algorithm that, for any PC^{X} instance with n terminal points in a metric space with doubling dimension at most k, runs in time*

$$n^{O(1)^{O(k)}} \cdot \exp(\sqrt{\log n} \cdot O(\frac{k}{\epsilon})^{O(k)}),$$

and returns a solution that is a $(1+\epsilon)$-approximation with constant probability.

Technical Issues. As a first trial, one might try to adapt the sparsity framework used in previous PTAS's for the TSP and Steiner forest problems [5, 12, 10] in doubling metrics. The framework typically uses a polynomial-time estimator H on any ball B, which gives a constant approximation for PC^{X} on some appropriately defined sub-instance around B. Intuitively, the estimator works because the local behavior of a (nearly) optimal solution can be well estimated by looking at the sub-instance locally. In particular, the following properties are needed in this framework:

- If $\mathsf{H}(B)$ is large, then the optimal solution for the sub-instance induced on B is large; moreover, any (nearly) optimal solution for the global instance would have a large part of its cost due to B.
- If $\mathsf{H}(B)$ is small, then for any (nearly) optimal solution F for the global instance, the cost of F contributed by the sub-instance due to B should be small.

While the first property is somehow straightforward, the following example shows that the second property is non-trivial to achieve in PC^{X}.

Example Instance: Figure 1. The example is defined on the real line. The terminals are grouped into two clusters. The left cluster contains $2m$ terminals, and the right cluster contains m terminals. Within each cluster, the distance between adjacent terminals is 1. The two clusters are at distance l apart. The penalty for each terminal is t. The parameters are chosen such that $l \gg mt$ and $t \gg m$. Observe that for PC^{X}, the optimal solution is to visit all the terminals in the left cluster with total edge weights $O(m)$ and incur the penalty mt for the terminals in the right cluster. The reason is that it will be too costly to add an edge to connect terminals from different clusters, and it is better to visit the cluster with more terminals and suffer the penalty for the cluster with fewer terminals.

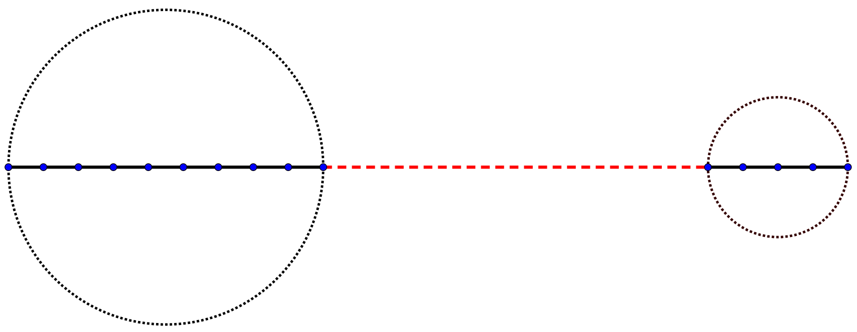


Figure 1 Example instance for PC^{X}.

Local Estimator Fails on the Example Instance. Suppose the estimator is applied around a ball B centered at some terminal in the right cluster with radius r. Then, any constant-approximate solution for the sub-instance needs to connect all $\Theta(r)$ terminals in the ball, since the penalty for any single terminal is too large. This costs $\Theta(r)$. However, in the optimal solution, no terminal in the right cluster is visited and all penalties are taken, which has cost $\Omega(tr)$. Hence, the estimator fails to serve as an upper bound for the contribution by ball B to the cost of an optimal solution.

The conclusion is that the optimal solution of a local sub-instance can differ a lot from how an optimal global solution behaves for that sub-instance.

Our Insight: Trading between Weight and Penalty. Our example in Figure 1 shows that what points a local optimal solution visits in a sub-instance can be very different from the points in the sub-instance visited by a global optimal solution. Our intuition is that the optimal cost of a sub-instance should reflect part of the cost in a global optimal solution due to the sub-instance. In other words, if a sub-instance has large optimal cost, then any global solution either (1) has a large weight within the sub-instance, or (2) suffers a large penalty due to unvisited terminals in the sub-instance. This insight leads to the following key ingredients to our solution.

1. **Inferring Local Behavior from Estimator.** In Lemma 5, we show that the value returned by the local estimator (which consists of both the weight and the penalty) on a ball B gives an upper bound on the weight $w(F|_B)$ of any (near) optimal solution F inside ball B. We emphasize that this estimator is an upper bound for the *weight* $w(F|_B)$ *only*, and is *not* an upper bound for both the weight and penalty of the optimal solution inside the ball. In the example in Figure 1, a global optimal solution does not visit the right cluster at all, and hence, the local estimator on the right cluster does give an upper bound on the *weight* part of the global solution due to the right cluster. This turns out to be sufficient because the sparsity of a solution is defined with respect to only the weight part (and not the penalty part).
   Hence, the local estimator can be used in the sparsity decomposition framework [5, 12, 10] to identify a *critical* instance W_1 (i.e., the local estimator reaches some threshold, but still not too large) around some ball B. Since the instance W_1 is sparse enough, an approximate solution F_1 can be obtained by the dynamic program framework. Then, one can recursively solve for an approximate solution F_2 for the remaining instance W_2. However, we need to carefully define W_2 and combine the solutions F_1 and F_2, because, as we remarked before, even if the approximate algorithm returns F_1 for the instance W_1, a near optimal global solution might not visit any terminals in W_1.

2. **Adaptive Recursion.** In all previous applications of the sparsity decomposition framework, after a critical ball B around some center u is identified, the original instance is decomposed into sub-instances W_1 and W_2 that can be solved independently.
   An issue in applying this framework is that after obtaining solutions F_1 and F_2 for the sub-instances, in the case that F_1 and F_2 are far away from each other as in our example in Figure 1 where it is too costly to connect them directly, it is not clear immediately which of F_1 and F_2 should be the weight part of the global solution and which would become the penalty part.
   We use a novel idea of the adaptive recursion, in which W_2 depends on the solution F_1 returned for W_1. The high level idea is that in defining the instance W_2, we add an extra terminal point at u, which becomes a representative for solution F_1. The penalty of u in W_2 is the sum of the penalties of terminals in W_1 minus the cost $c(F_1)$ of solution F_1. After a solution F_2 for W_2 is returned, if F_2 does not visit the terminal u, then edges in F_1 are discarded, otherwise the edges in F_1 and F_2 are combined to return a global solution. We can see that in either case, the sum $c(F_1) + c(F_2)$ of the costs of the two solutions reflect the cost of the global solution. In the first case, F_2 does not visit u and hence, $c(F_2)$ contains the penalty due to u, which is the penalties of unvisited terminals in W_1 minus $c(F_1)$. Therefore, when $c(F_1)$ is added back, the sum simply contains the original penalties of unvisited terminals in W_1.
   In the second case, F_2 does visit u and does not incur a penalty due to u. Therefore, $c(F_1) + c(F_2)$ does reflect the cost of the global solution after combining F_1 and F_2.

Revisiting the Sparsity Structural Lemma. Many PTAS's in the literature for TSP-like problems in doubling metrics rely on the sparsity structural lemma [5, Lemma 3.1]. Intuitively, it says that if a solution is sparse, then there exists a structurally light solution that is $(1 + \epsilon)$-approximate. Hence, one can restrict the search space to structurally light solutions, which can be explored by a dynamic program algorithm. Because of the significance of this lemma, we believe that it is worthwhile to give it a more formal inspection, and in particular, resolve some significant technical issues as follows.

- *Issue with Conditioning on the Randomness of Hierarchical Decomposition.* Given a hierarchical decomposition and a solution T, the first step is to reroute the solution such that every cluster is only visited through some designated points known as *portals*. The randomness in the hierarchical decomposition is used to argue that the expected increase in cost to make the solution portal-respecting is small.
  However, typically the randomness in the hierarchical decomposition is still needed in subsequent arguments. Hence, if one analyzes the portal-respecting procedure as a conceptually separate step, then subsequent uses of the randomness of the hierarchical decomposition need to condition on the event that the portal-respecting step does not increase the cost too much. Moreover, edges added in the portal-respecting step are actually random objects depending on the hierarchical decomposition, and hence, will in fact cross some clusters with probability 1. Unfortunately, even in the original paper by Talwar [20] on the QPTAS for TSP in doubling metrics, these issues were not addressed properly.
- *Issues with Patching Procedure.* A patching procedure is typically used to reduce the number of times a cluster is crossed. In the literature, after reducing the number of crossings, the triangle inequality is used to implicitly add some shortcutting edges outside the cluster. However, it is never argued whether these new shortcutting edges are still portal-respecting. It is plausible that making them portal-respecting might introduce new crossings.

From the above discussion, it is evident that one should consider the portal-respecting step and the patching procedure together, because they both rely on the randomness of the hierarchical decomposition. To make our arguments formal, we need a more precise notation to describe portals, and we actually revisit the whole randomized hierarchical decomposition to make all relevant definitions precise. We analyze the portal-respecting step and the patching procedure together through a sophisticated accounting argument so that the patching cost is eventually charged back to the original solution (as opposed to stopping at the transformed portal-respecting solution).

Moreover, we give a unified patching lemma that works for both $\mathsf{PC}^{\mathsf{TSP}}$ and $\mathsf{PC}^{\mathsf{STP}}$. Even though our proofs use similar ideas as previous works, the charging argument is significantly different. Specifically, our argument does not rely on the small MST lemma [20, Lemma 6], which was also used in [5].

Paper Organization. Section 2 gives the formal notation and describes the outline of the sparsity decomposition framework to solve PC^{X}. Section 3 gives the properties of the local sparsity estimator. Section 4 gives the technical details of the sparsity decomposition and shows that approximate solutions in sub-instances can be combined to give a good approximation to the global instance. Some proofs, together with other sections, are omitted due to space limit, and they can be found in the full version [11].

2 Preliminaries

We consider a metric space $M = (X, d)$ (see [15, 19] for more details on metric spaces), where we refer to an element $x \in X$ as a point or a vertex. For $x \in X$ and $\rho \geq 0$, a *ball* $B(x, \rho)$ is the set $\{y \in X \mid d(x, y) \leq \rho\}$. The *diameter* $\mathsf{Diam}(Z)$ of a set $Z \subset X$ is the maximum distance between points in Z. For $S, T \subset X$, we denote $d(S, T) := \min\{d(x, y) : x \in S, y \in T\}$, and for $u \in X$, $d(u, T) := d(\{u\}, T)$. Given a positive integer m, we denote $[m] := \{1, 2, \ldots, m\}$.

A set $S \subseteq X$ is a ρ-packing, if any two distinct points in S are at a distance more than ρ away from each other. A set S is a ρ-cover for $Z \subseteq X$, if for any $z \in Z$, there exists $x \in S$ such that $d(x, z) \leq \rho$. A set S is a ρ-net for Z, if S is a ρ-packing and a ρ-cover for Z. We assume the access to an oracle that takes a series of balls $\{B_i\}_i$ where each B_i is identified by the center and radius, and returns a point $x \in X$ such that $\forall i, x \notin S_i$[4]. A greedy algorithm can construct a ρ-net efficiently given the access to this oracle.

We consider metric spaces with *doubling dimension* [3, 18] at most k; this means that for all $x \in X$, for all $\rho > 0$, every ball $B(x, 2\rho)$ can be covered by the union of at most 2^k balls of the form $B(z, \rho)$, where $z \in X$. The following fact captures a standard property of doubling metrics.

▶ **Fact 2** (Packing in Doubling Metrics [18]). *Suppose in a metric space with doubling dimension at most k, a ρ-packing S has diameter at most R. Then, $|S| \leq (\frac{2R}{\rho})^k$.*

Edges. An edge[5] e is an unordered pair $e = \{x, y\} \in \binom{X}{2}$ whose weight $w(e) = d(x, y)$ is induced by the metric space (X, d). Given a set F of edges, its vertex set $V(F) := \cup_{e \in F}\, e \subset X$ is the vertices covered (or *visited*) by the edges in F. If $T \subset X$ is a set of vertices, we use the shorthand $T \setminus F := T \setminus V(F)$ to denote the vertices in T that are not covered by F.

[4] Such an oracle is trivial to construct for finite metric spaces. It may also be efficiently constructed for many special infinite metric spaces, such as bounded dimensional Euclidean spaces.

[5] To have a complete description, we also need the notion of self-loop, which is simply a singleton $\{x\}$.

Problem Definition. We give a unifying framework for the prize collecting traveling salesman problem ($\mathsf{PC}^{\mathsf{TSP}}$) and the prize collecting Steiner tree problem ($\mathsf{PC}^{\mathsf{STP}}$), and we use PC^{X} when the description applies to both problems. An instance $W = (T, \pi)$ of PC^{X} consists of a set $T \subset X$ of *terminals* (where $|W| := |T| = n$) and a penalty function $\pi : T \to \mathbb{R}_+$. The goal is to find a (multi-)set $F \subset \binom{X}{2}$ of edges with minimum cost[6] $c_W(F) := w(F) + \pi(T \setminus F)$, such that the following additional conditions are satisfied for each specific problem:

- For $\mathsf{PC}^{\mathsf{TSP}}$, the edges in the multi-set F form a circuit on $V(F)$; for $|V(F)| = 1$, F contains only a single self-loop (with zero weight).
- For $\mathsf{PC}^{\mathsf{STP}}$, the edges F form a connected graph on $V(F)$, where we also allow the degenerate case when F is a singleton containing a self-loop. The vertices in $V(F) \setminus T$ are known as *Steiner* points.

Simplifying Assumptions and Rescaling Instance. Fix some constant $\epsilon > 0$. Since we consider asymptotic running time to obtain $(1+\epsilon)$-approximation for PC^{X}, we consider sufficiently large $n > \frac{1}{\epsilon}$. Since F can contain a self-loop, an optimal solution covers at least one terminal u. Moreover, there is some terminal v (which could be the same as u) such that the solution covers v and does not cover any terminal v' with $d(u, v') > d(u, v)$. Since we aim for polynomial time algorithms, we can afford to enumerate the $O(n^2)$ choices for u and v.

For some choice of u and v, suppose $R := d(u, v)$. Then, R is a lower bound on the cost of an optimal solution. Moreover, the optimal solution F has weight $w(F)$ at most nR, and hence, we do not need to consider points at distances larger than nR from u. Since F contains at most $2n$ edges (because of Steiner points in $\mathsf{PC}^{\mathsf{STP}}$), if we consider an $\frac{\epsilon R}{32n^2}$-net S for X and replace every point in F with its closest net-point in S, the cost increases by at most $\epsilon \cdot \mathsf{OPT}$. Hence, after rescaling, we can assume that inter-point distance is at least 1 and we consider distances up to $O(\frac{n^3}{\epsilon}) = \text{poly}(n)$. By the packing property of doubling dimension (Fact 2), we can hence assume $|X| \leq O(\frac{n}{\epsilon})^{O(k)} \leq O(n)^{O(k)}$.

Hierarchical Nets. As in [5], we consider some parameter $s = (\log n)^{\frac{c}{k}} \geq 4$, where $0 < c < 1$ is a universal constant that is sufficiently small. Set $L := O(\log_s n) = O(\frac{k \log n}{\log \log n})$. A greedy algorithm can construct $N_L \subseteq N_{L-1} \subseteq \cdots \subseteq N_1 \subseteq N_0 = N_{-1} = \cdots = X$ such that for each i, N_i is an s^i-net for N_{i-1}, where we say *distance scale* s^i is of *height* i.

Net-Respecting Solution. As defined in [5], a graph F is net-respecting with respect to $\{N_i\}_{i \in [L]}$ and $\epsilon > 0$ if for every edge $\{x, y\}$ in F, both x and y belong to N_i, where $s^i \leq \epsilon \cdot d(x, y) < s^{i+1}$. By [5, Lemma 1.6], any graph F may be converted to a net-respecting F' visiting all points that F visits, and $w(F') \leq (1 + O(\epsilon)) \cdot w(F)$.

Given an instance W of a problem, let $\mathsf{OPT}(W)$ be an optimal solution; when the context is clear, we also use $\mathsf{OPT}(W)$ to denote the cost $c(\mathsf{OPT}(W))$, which includes both its weight and the incurred penalty; similarly, $\mathsf{OPT}^{nr}(W)$ refers to an optimal net-respecting solution.

2.1 Overview

We achieve a PTAS for PC^{X} by a unified framework, which is based on the framework of sparse instance decomposition as in [5, 12, 10].

[6] When the context is clear, we drop the subscript in $c_W(\cdot)$.

Sparse Solution [5]. Given an edge set F and a subset $S \subseteq X$, $F|_S := \{e \in F : e \subseteq S\}$ is the edges in F totally contained in S. An edge set F is called q-sparse, if for all $i \in [L]$ and all $u \in N_i$, $w(F|_{B(u,3s^i)}) \leq q \cdot s^i$.

Sparsity Structural Property. An important technical lemma [5, Lemma 3.1] in this framework states that if a (net-respecting) solution F is sparse, then with constant probability, there is some $(1+\epsilon)$-approximate solution $\widehat{F}$ that is *structurally light* with respect to some randomized *hierarchical decomposition.* Then, a bottom-up dynamic program based on the hierarchical decomposition searches for the best solution with the lightness structural property in polynomial time.

▶ Remark. We observe that this technical lemma is used crucially in all previous works on PTAS's for TSP variants in doubling metrics. Hence, we believe that its proof should be verified rigorously. In Section 1, we outlined the technical issue in the original proof [5], and this issue actually appeared as far as in the first paper on TSP for doubling metrics [20]. In the full version, we give a detailed description to complete the proof of this important lemma.

Sparsity Heuristic. As in [5, 12, 10], we estimate the local sparsity of an optimal net-respecting solution with a heuristic. For $i \in [L]$ and $u \in N_i$, given an instance W, the heuristic $\mathsf{H}_u^{(i)}(W)$ is supposed to estimate the sparsity of an optimal net-respecting solution in the ball $B' := B(u, O(s^i))$. We shall see in Section 3 that the heuristic actually gives a constant approximation to some appropriately defined sub-instance W' in the ball B'.

Divide and Conquer. Once we have a sparsity estimator, the original instance can be decomposed into sparse sub-instances, whose approximate solutions can be found efficiently. As we shall see, the partial solutions are combined with the following extension operator. The algorithm outline is described in Figure 2.

▶ **Definition 3** (Solution Extension). Given two partial solutions F and F' of edges, we define the *extension* of F with F' at point u as $F \hookleftarrow_u F' := \begin{cases} F \cup F', & \text{if } u \in V(F) \cap V(F'); \\ F, & \text{otherwise.} \end{cases}$

Analysis of Approximation Ratio. We follow the inductive proof as in [5] to show that with constant probability (where the randomness comes from DP), $\mathsf{ALG}(W)$ in Figure 2 returns a solution with expected length at most $\frac{1+\epsilon}{1-\epsilon} \cdot \mathsf{OPT}^{nr}(W)$, where expectation is over the randomness of decomposition into sparse instances in Step 4.

As we shall see, in $\mathsf{ALG}(W)$, the subroutine DP is called at most poly(n) times (either explicitly in the recursion or in the heuristic $\mathsf{H}^{(i)}$). Hence, with constant probability, all solutions returned by all instances of DP have appropriate approximation guarantees.

Suppose F_1 and F_2 are solutions returned by $\mathsf{DP}(W_1)$ and $\mathsf{ALG}(W_2)$, respectively. We use c_i as a shorthand for c_{W_i}, for $i = 1, 2$, and c as a shorthand for c_W. Since we assume that W_1 is sparse enough and DP behaves correctly, $c_1(F_1) \leq (1+\epsilon) \cdot \mathsf{OPT}(W_1)$. The induction hypothesis states that $\mathbf{E}[c_2(F_2)|W_2] \leq \frac{1+\epsilon}{1-\epsilon} \cdot \mathsf{OPT}^{nr}(W_2)$.

In Step 4, equation (2) guarantees that $\mathbf{E}[\mathsf{OPT}(W_1)] \leq \frac{1}{1-\epsilon} \cdot (\mathsf{OPT}^{nr}(W) - \mathbf{E}[\mathsf{OPT}^{nr}(W_2)])$. By equation (1), $c(F_2 \hookleftarrow_u F_1) \leq c_1(F_1) + c_2(F_2)$. Hence, it follows that

$$\mathbf{E}[\mathsf{ALG}(W)] \leq \mathbf{E}[c_1(F_1) + c_2(F_2)] \leq \frac{1+\epsilon}{1-\epsilon} \cdot \mathsf{OPT}^{nr}(W) = (1 + O(\epsilon)) \cdot \mathsf{OPT}(W),$$

achieving the desired ratio.

Generic Algorithm. We describe a generic framework that applies to PC^{X}. Similar framework is also used in [5, 12, 10] to obtain PTAS's for TSP related problems. Given an instance W, we describe the recursive algorithm $\mathsf{ALG}(W)$ as follows. This description is mostly the same with that in [10], except that the decomposition in Step 4 is more involved.

1. **Base Case.** If $|W| = n$ is smaller than some constant threshold, solve the problem by brute force, recalling that $|X| \le O(\frac{n}{\epsilon})^{O(k)}$.
2. **Sparse Instance.** If for all $i \in [L]$, for all $u \in N_i$, $\mathsf{H}_u^{(i)}(W)$ is at most $q_0 \cdot s^i$, for some appropriate threshold q_0, call the subroutine $\mathsf{DP}(W)$ to return a solution, and terminate.
3. **Identify Critical Instance.** Otherwise, let i be the smallest height such that there exists $u \in N_i$ with *critical* $\mathsf{H}_u^{(i)}(W) > q_0 \cdot s^i$; in this case, choose $u \in N_i$ such that $\mathsf{H}_u^{(i)}(W)$ is maximized.
4. **Divide and Conquer.** Define a sub-instance W_1 from around the critical instance (possibly using randomness). Loosely speaking, W_1 is a sparse enough sub-instance induced in the region around u at distance scale s^i. Since it is sparse enough, we apply the dynamic programming algorithm on W_1 and get solution F_1.
   We define an appropriate sub-instance W_2 *with the information of* F_1. Intuitively, W_2 captures the remaining sub-problem not included in W_1. We emphasize that as opposed to previous work [5, 12, 10], W_2 can depend on F_1 (through the choice of the penalty function). Moreover, we ensure that any solution F_2 of W_2 can be extended to $F_2 \hookleftarrow_u F_1$ as a solution for W, and the following holds:
$$c_W(F_2 \hookleftarrow_u F_1) \le c_{W_1}(F_1) + c_{W_2}(F_2). \tag{1}$$
   We solve W_2 recursively and suppose the solution is F_2. We note that $\mathsf{H}_u^{(i)}(W_2) \le q_0 \cdot s^i$, and hence the recursion will terminate.
   Moreover, the following property holds:
$$\mathbf{E}[\mathsf{OPT}(W_1)] \le \frac{1}{1-\epsilon} \cdot (\mathsf{OPT}^{nr}(W) - \mathbf{E}[\mathsf{OPT}^{nr}(W_2)]), \tag{2}$$
   where the expectation is over the randomness of the decomposition.
   We return $F := F_2 \hookleftarrow_u F_1$ as a solution to W.

Figure 2 Algorithm Outline.

Analysis of Running Time. As mentioned above, if $\mathsf{H}_u^{(i)}(W)$ is found to be critical, then in the decomposed sub-instances W_1 and W_2, $\mathsf{H}_u^{(i)}(W_2)$ should be small. Hence, it follows that there will be at most $|X| \cdot L = \text{poly}(n)$ recursive calls to ALG. Therefore, as far as obtaining polynomial running times, it suffices to analyze the running time of the dynamic program DP. The details are provided in the full version.

3 Sparsity Estimator for PC^{X}

Recall that in the framework outlined in Section 2, given an instance W of PC^{X}, we wish to estimate the weight of $\mathsf{OPT}^{nr}(W)|_{B(u,3s^i)}$ with some heuristic $\mathsf{H}_u^{(i)}(W)$. We consider a more general sub-instance associated with the ball $B(u, ts^i)$ for $t \ge 1$.

Auxiliary Sub-Instance. Given an instance $W = (T, \pi)$, $i \in [L]$, $u \in N_i$ and $t \geq 1$, the sub-instance $W_u^{(i,t)}$ is characterized by terminal set $W \cap B(u, ts^i)$, equipped with penalties given by the same π. Using the classical (deterministic) 2-approximation algorithms by Goemans and Williamson for PC^{X} [17], we obtain a 2-approximation and then make it net-respecting to produce solution $F_u^{(i,t)}$, which has cost $c(F_u^{(i,t)}) \leq 2(1+O(\epsilon)) \cdot \mathsf{OPT}(W_u^{(i,t)})$.

Defining the Heuristic. The heuristic is defined as $\mathsf{H}_u^{(i)}(W) := c(F_u^{(i,4)})$.

In order to show that the heuristic gives a good upper bound on the local sparsity of an optimal net-respecting solution, we need the following structural result in Proposition 4 [10, Lemma 3.2] on the existence of long chain in well-separated terminals in a Steiner tree. As we shall see, the corresponding argument for the case $\mathsf{PC}^{\mathsf{TSP}}$ is trivial.

Given an edge set F, a *chain* in F is specified by a sequence of points $(p_1, p_2, \dots, p_l)$ such that there is an edge $\{p_i, p_{i+1}\}$ in F between adjacent points, and the degree of an internal point p_i (where $2 \leq i \leq l-1$) in F is exactly 2.

▶ **Proposition 4** (Well-Separated Terminals Contains A Long Chain). *Suppose S and T are sets in a metric space with doubling dimension at most k such that $\mathsf{Diam}(S \cup T) \leq D$, and $d(S,T) \geq \tau D$, where $0 < \tau < 1$. Suppose F is an optimal net-respecting Steiner tree covering the terminals in $S \cup T$. Then, there is a chain in F with weight at least $\frac{\tau^2}{4096k^2} \cdot D$ such that any internal point in the chain is a Steiner point.*

▶ **Lemma 5** (Local Sparsity Estimator). *Let F be an optimal net-respecting solution for an instance W of PC^{X}. Then, for any $i \in [L]$, $u \in N_i$ and $t \geq 1$, we have*
$w(F|_{B(u,ts^i)}) \leq c(F_u^{(i,t+1)}) + O(\frac{skt}{\epsilon})^{O(k)} \cdot s^i$.

Proof. We follow the proof strategy in [10, Lemma 3.3], except that now a feasible solution needs not visit all terminals and can incur penalties instead. We denote $B := B(u, ts^i)$ and $\widehat{B} := B(u, (t+1)s^i)$.

Given an optimal net-respecting solution F for instance W of PC^{X}, we shall construct another net-respecting solution in the following steps.

1. Remove edges in $F|_B$.
2. Add edges $F_u^{(i,t+1)}$ corresponding to some approximate solution to the instance $W_u^{(i,t+1)}$ restricted to the ball $\widehat{B}$.
3. Let $\eta := \Theta(\frac{\epsilon}{(t+1)k^2})$, where the constant in Theta depends on Proposition 4. Let j be the integer such that $s^j \leq \max\{1, \Theta(\frac{\epsilon}{(t+1)k^2}) \cdot s^i\} < s^{j+1}$.
   Add edges in a minimum spanning tree H of $N_j \cap B(u, (t+2)s^i)$ and edges to connect H to $F_u^{(i,t+1)}$.
   Convert each added edge into a net-respecting path if necessary. Observe that the weight of edges added in this step is $O(\frac{stk}{\epsilon})^{O(k)} \cdot s^i$.
4. So far we have accounted for every terminal inside $\widehat{B}$, which is either visited or charged with its penalty according to $c(F_u^{(i,t+1)})$. We will give a more detailed description to ensure that the terminals outside $\widehat{B}$ that are covered by F will still be covered by the new solution.
   For $\mathsf{PC}^{\mathsf{TSP}}$, we will show that this step can be achieved by increasing the weight by at most $O(\frac{stk}{\epsilon})^{O(k)} \cdot s^i$; for $\mathsf{PC}^{\mathsf{STP}}$, this can be achieved by replacing some edges without increasing the weight.

Hence, after the claim in Step 4 is proved, the optimality of F implies the result.

Ensuring Terminals Outside $\widehat{B}$ are accounted for. We achieve this by considering the following steps.

1. Consider a connected component C in $F \setminus (F|_B)$. Recall that the goal is to make sure that all terminals outside $\widehat{B}$ that are visited by C will also be visited in the new solution.
2. Pick some x in $C \cap B$. If no such x exists, this implies that we have the trivial situation $F|_B = \emptyset$. Let $\widehat{C} \subseteq C$ be the maximal connected component containing x that is contained within $\widehat{B}$. Define $S := \widehat{C} \cap B$ (which contains x) and $T := \{y \in \widehat{C} \cap \widehat{B} : \exists v \notin \widehat{B}, \{y, v\} \in F\}$, which corresponds to the points that are connected to the outside $\widehat{B}$. Again, the case that $T = \emptyset$ is trivial.

Case (a): There exists $y \in T$, $d(u, y) \leq (t + \frac{1}{2})s^i$. In this case, this implies there is some $v \notin \widehat{B}$ such that $\{y, v\} \in F$ and $d(y, v) \geq \frac{s^i}{2}$. Since F is net-respecting, this implies that $y \in N_j$ and hence, the component $\widehat{C}$ (and also C) is already connected to H.

Case (b): For all $y \in T$, $d(u, y) > (t + \frac{1}{2})s^i$. We next show that there is a long chain contained in $\widehat{C}$. For $\mathsf{PC}^{\mathsf{TSP}}$, this is trivial, because we know that T contains only y, and $\widehat{C}$ is a chain from $a = x$ to $b = y$ of length at least $d(x, y) \geq \frac{s^i}{2}$.

For $\mathsf{PC}^{\mathsf{STP}}$, by the optimality of F, it follows that $\widehat{C}$ is an optimal net-respecting Steiner tree covering vertices in $S \cup T$. Hence, using Proposition 4, $\widehat{C}$ contains some chain from a to b with length at least $4\eta s^i$ (where the constant in the Theta in the definition of η is chosen such that this holds).

Once we have found this chain from a to b, we remove the edges in this chain. Hence, we can use this extra weight to connect a and b to their corresponding closest points in N_j via a net-respecting path; observe that for $\mathsf{PC}^{\mathsf{TSP}}$, it suffices to connect only $b = y$ to it closest point in N_j.

Finally, observe that for $\mathsf{PC}^{\mathsf{TSP}}$, it is possible to carry out the above procedures such that all vertices with odd degrees are in the minimum spanning tree H. Therefore, extra edges are added to ensure that the degree of every vertex is even to ensure the existence of an Euler circuit. This has extra cost at most $w(H) \leq O(\frac{stk}{\epsilon})^{O(k)} \cdot s^i$. This completes the proof. ◀

▶ **Corollary 6** (Threshold for Critical Instance). *Suppose F is an optimal net-respecting solution for an instance W of PC^{X}, and $q \geq \Theta(\frac{sk}{\epsilon})^{\Theta(k)}$. If for all $i \in [L]$ and $u \in N_i$, $\mathsf{H}_u^{(i)}(W) \leq qs^i$, then F is $2q$-sparse.*

4 Decomposition into Sparse Instances

In Section 3, we define a heuristic $\mathsf{H}_u^{(i)}(W)$ to detect a critical instance around some point $u \in N_i$ at distance scale s^i. We next describe how the instance W of PC^{X} can be decomposed into W_1 and W_2 such that equations (1) and (2) in Section 2.1 are satisfied.

Decomposing a Critical Instance. We define a threshold $q_0 := \Theta(\frac{sk}{\epsilon})^{\Theta(k)}$ according to Corollary 6. As stated in Section 2.1, a critical instance is detected by the heuristic when a smallest $i \in [L]$ is found for which there exists some $u \in N_i$ such that $\mathsf{H}_u^{(i)}(W) = c(F_u^{(i,4)}) > q_0 s^i$. Moreover, in this case, $u \in N_i$ is chosen to maximize $\mathsf{H}_u^{(i)}(W)$. To achieve a running time with an $\exp(O(1)^{k \log(k)})$ dependence on the doubling dimension k, we also apply the technique in [12] to choose the cutting radius carefully.

▶ **Claim 7** (Choosing Radius of Cutting Ball). *Denote $\mathsf{T}(\lambda) := c(F_u^{(i,4+2\lambda)})$. Then, there exists $0 \leq \lambda < k$ such that $\mathsf{T}(\lambda + 1) \leq 30k \cdot \mathsf{T}(\lambda)$.*

Proof. The proof is omitted and can be found in the full version. ◀

Cutting Ball and Sub-Instances. Suppose $\lambda \geq 0$ is picked as in Claim 7, and sample $h \in [0, \frac{1}{2}]$ uniformly at random. Define $B := B(u, (4 + 2\lambda + h)s^i)$. The original instance $W = (T, \pi)$ is decomposed into instances W_1 and W_2 as follows:

- For $W_1 = (T_1, \pi_1)$, the terminal set is $T_1 := (B \cap T) \cup \{u\}$, where for $v \neq u$ $\pi_1(v) := \pi(v)$ and $\pi_1(u) := +\infty$. We denote the cost function associated with W_1 by c_1.
- Suppose F_1 is the (random) solution for instance W_1 (that covers u) returned by the dynamic program for sparse instances (which can be found in the full version). Then, instance $W_2 = (T_2, \pi_2)$ is defined with respect to F_1. The terminal set is $T_2 := (T \backslash B) \cup \{u\}$. For $v \in T_2 \setminus \{u\}$, $\pi_2(v) := \pi(v)$ is the same; however, $\pi_2(u) := \pi(T \cap B) - c_1(F_1) = \pi(T \cap B \cap F_1) - w(F_1)$.

  Observe that the instance W_2 depends on F_1 through the choice of the penalty for u.

▶ **Lemma 8** (Combining Solutions of Sub-Instances)**.** *Suppose instance W_1 is defined with cost function c_1 and instance W_2 is defined with respect to F_1 of W_1. Furthermore, suppose $\widehat{F}_2$ is a solution to instance W_2, whose cost function is denoted as c_2. Then, we have the following.*

(i) *Suppose $\widehat{F}_1$ is any solution to W_1 that contains u, and let $F := \widehat{F}_2 \hookleftarrow_u \widehat{F}_1$. If $\widehat{F}_2$ covers u, then $F = \widehat{F}_2 \cup \widehat{F}_1$ is a solution to W with cost $c(F) \leq c_1(\widehat{F}_1) + c_2(\widehat{F}_2)$; if F_2 does not cover u, then $F = \widehat{F}_2$ is a solution to W with cost $c(F) \leq c_1(F_1) + c_2(\widehat{F}_2)$. This implies (1) in Section 2.1.*

(ii) *The sub-instance W_2 does not have a critical instance with height less than i, and $\mathsf{H}_u^{(i)}(W_2) = 0$.*

(iii) $\mathsf{H}_u^{(i)}(W_1) \leq O(s)^{O(k)} \cdot q_0 \cdot s^i$.

Proof. The proof is omitted and can be found in the full version. ◀

▶ **Lemma 9** (Combining Costs of Sub-Instances)**.** *Suppose F is an optimal net-respecting solution for instance W of* PC^{X}*. Then, for any realization of the decomposed sub-instances W_1 and W_2 as described above, there exist (not necessarily net-respecting) solution $\widehat{F}_1$ for W_1 and net-respecting solution $\widehat{F}_2$ for W_2 such that $(1 - \epsilon) \cdot \mathbf{E}\left[c_1(\widehat{F}_1)\right] + \mathbf{E}\left[c_2(\widehat{F}_2)\right] \leq c_W(F)$, where the expectation is over the randomness to generate W_1 and W_2. Recall that the randomness to generate W_1 and W_2 involves the random ball B and the randomness used in the dynamic program to generate F_1 to produce instance W_2 and its cost function c_2.*

Proof. The proof is omitted and can be found in the full version. ◀

References

1 Aaron Archer, MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Howard J. Karloff. Improved approximation algorithms for prize-collecting steiner tree and TSP. *SIAM J. Comput.*, 40(2):309–332, 2011.

2 Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.

3 P. Assouad. Plongements lipschitziens dans $\mathbf{R}^n$. *Bull. Soc. Math. France*, 111(4):429–448, 1983.

4 Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

5 Yair Bartal, Lee-Ad Gottlieb, and Robert Krauthgamer. The traveling salesman problem: Low-dimensionality implies a polynomial time approximation scheme. *SIAM J. Comput.*, 45(4):1563–1581, 2016.

6 M Bateni, Chandra Chekuri, Alina Ene, Mohammad Taghi Hajiaghayi, Nitish Korula, and Dániel Marx. Prize-collecting steiner problems on planar graphs. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1028–1049. Society for Industrial and Applied Mathematics, 2011.

7 MohammadHossein Bateni and MohammadTaghi Hajiaghayi. Euclidean prize-collecting steiner forest. *Algorithmica*, 62(3-4):906–929, 2012.

8 Daniel Bienstock, Michel X Goemans, David Simchi-Levi, and David Williamson. A note on the prize collecting traveling salesman problem. *Mathematical programming*, 59(1):413–420, 1993.

9 Glencora Borradaile, Philip N. Klein, and Claire Mathieu. A polynomial-time approximation scheme for euclidean steiner forest. *ACM Trans. Algorithms*, 11(3):19:1–19:20, 2015.

10 T.-H. Hubert Chan, Shuguang Hu, and Shaofeng H.-C. Jiang. A PTAS for the steiner forest problem in doubling metrics. In *FOCS*, pages 810–819. IEEE Computer Society, 2016.

11 T.-H. Hubert Chan, Haotian Jiang, and Shaofeng H.-C. Jiang. A unified PTAS for prize collecting TSP and steiner tree problem in doubling metrics. *CoRR*, abs/1710.07774, 2017.

12 T.-H. Hubert Chan and Shaofeng H.-C. Jiang. Reducing curse of dimensionality: Improved PTAS for TSP (with neighborhoods) in doubling metrics. In *SODA*, pages 754–765. SIAM, 2016.

13 Kenneth L. Clarkson. Nearest neighbor queries in metric spaces. *Discrete & Computational Geometry*, 22(1):63–93, 1999. doi:10.1007/PL00009449.

14 Erik D Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Contraction decomposition in h-minor-free graphs and algorithmic applications. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 441–450. ACM, 2011.

15 M. M. Deza and M. Laurent. *Geometry of cuts and metrics*, volume 15 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 1997.

16 Michel X Goemans. Combining approximation algorithms for the prize-collecting tsp. *arXiv preprint arXiv:0910.0553*, 2009.

17 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.

18 Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *FOCS*, pages 534–543. IEEE Computer Society, 2003.

19 J. Matoušek. *Lectures on discrete geometry*, volume 212 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2002.

20 Kunal Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In *STOC*, pages 281–290. ACM, 2004.

Near-Optimal Distance Emulator for Planar Graphs

Hsien-Chih Chang[1]
University of Illinois at Urbana-Champaign, USA
hchang17@illinois.edu

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

Shay Mozes[2]
IDC Herzliya, Israel
smozes@idc.ac.il

Oren Weimann[3]
University of Haifa, Israel
oren@cs.haifa.ac.il

Abstract

Given a graph G and a set of terminals T, a *distance emulator* of G is another graph H (not necessarily a subgraph of G) containing T, such that all the pairwise distances in G between vertices of T are preserved in H. An important open question is to find the smallest possible distance emulator.

We prove that, given any subset of k terminals in an n-vertex undirected unweighted planar graph, we can construct in $\tilde{O}(n)$ time a distance emulator of size $\tilde{O}(\min(k^2, \sqrt{k \cdot n}))$. This is optimal up to logarithmic factors. The existence of such distance emulator provides a straightforward framework to solve distance-related problems on planar graphs: Replace the input graph with the distance emulator, and apply whatever algorithm available to the resulting emulator. In particular, our result implies that, on any unweighted undirected planar graph, one can compute all-pairs shortest path distances among k terminals in $\tilde{O}(n)$ time when $k = O(n^{1/3})$.

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

Keywords and phrases planar graphs, shortest paths, metric compression, distance preservers, distance emulators, distance oracles

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.16

Related Version A full version of the paper is available at `https://arxiv.org/abs/1807.01478`.

Acknowledgements The authors would like to thank Timothy Chan, Jeff Erickson, Sariel Har-Peled, and Yipu Wang for helpful discussions. The first author express special thanks to 施鴻逸 (Hong-Yi Shih) for discussion back in NTU in the early days that made this paper possible.

[1] Supported partially by NSF grant CCF-1408763.
[2] Supported partially by ISF grants 794/13 and 592/17
[3] Supported partially by ISF grants 794/13 and 592/17

© Hsien-Chih Chang, Paweł Gawrychowski, Shay Mozes, and Oren Weimann;
licensed under Creative Commons License CC-BY

1 Introduction

The planar graph metric is one of the most well-studied metrics in graph algorithms, optimizations, and computer science in general. The planar graph *metric compression* problem is to compactly represent the distances among a subset T of k vertices, called **terminals**, in an n-vertex planar graph G. Without compression (and when G is unweighted) these distances can be naïvely represented with $O(\min(k^2 \log n, n))$ bits by either explicitly storing the $k \times k$ distances or alternatively by storing the entire graph (naïvely, this takes $O(n \log n)$ bits, but can be done with $O(n)$ bits [60, 55, 24, 13]).

A natural way to compress G is to replace it by another graph H that contains T as vertices, and the distances between vertices in T are preserved in H. In other words, $d_G(x, y) = d_H(x, y)$ holds for every pair of vertices x and y in T. Such graph H is called a **distance emulator** of G with respect to T (or a **distance preserver** in the case where H is required to be a subgraph of G). Distance emulators are appealing algorithmically, since we can readily feed them into our usual graph algorithms. They have been studied as compact representations of graphs [16, 26, 25, 15, 14, 8], and used as fundamental building blocks in algorithms and data structures for distance-related problems [16, 28, 31, 1, 2, 3, 42]. Similar concepts like additive and multiplicative *spanners* [28, 40, 12, 57, 63, 11] and *distance labelings* [39, 62, 53, 45, 34, 9, 35, 10] are popular topics with abundant results.

In planar graphs, an extensive study has been done for the case where the emulator H is required to be a minor of G [40, 21, 30, 11, 32, 23, 38, 37, 51]. Restricting H to be a minor of G has proven useful when H is only required to *approximate* the distances between vertices in T, say, up to a $(1 + \varepsilon)$ multiplicative error. For exact distances however, Krauthgamer, Nguyen, and Zondiner [50] have shown a lower bound of $\Omega(k^2)$ on the size of H, even when G is an unweighted grid graph and H is a (possibly weighted) minor emulator.

In general, an emulator H does not have to be a subgraph or a minor of G. In fact, H can be non-planar even when G is planar. Even in this setting, for *weighted* planar graphs, we cannot beat the naïve bound because one can encode an arbitrary $k \times k$ binary matrix using subset distances among $2k$ vertices in a weighted planar graph [34, 4]. Since we cannot compress an arbitrary binary $k \times k$ matrix into less than k^2 bits, we again have an $\Omega(k^2)$ lower bound.

What about *unweighted* planar graphs? Various distance-related problems in unweighted graphs enjoy algorithms and techniques [28, 49, 12, 57, 63, 64, 61, 22, 65, 29] which outperform their weighted counterparts. Indeed, for the metric compression problem in unweighted planar graphs, Abboud, Gawrychowski, Mozes, and Weimann [4] have very recently shown that we can in fact beat the $O(k^2)$ bound. They showed that the distances between any k terminals can be encoded using $\tilde{O}(\min(k^2, \sqrt{k \cdot n}))$ bits[4]. The encoding of Abboud et al. is optimal up to logarithmic factors. However, it is not an emulator! Goranci, Henzinger, and Peng [37] raised the question of whether such encoding can be achieved by an emulator. We answer this question in the affirmative.

Our results. We show that the encoding of Abboud et al. can be turned into an emulator, with no asymptotic overhead. Namely, we prove that any unweighted planar graph has a near-optimal size distance emulator.

[4] The $\tilde{O}(\cdot)$ notation hides polylogarithmic factors in n.

▶ **Theorem 1.** *Let G be an n-vertex undirected unweighted planar graph, and let T be the set of k terminals. A directed weighted graph H of size $O(\min(k^2, \sqrt{k \cdot n} \log^3 n))$ can be constructed in $O(n \log^4 n)$ time as a distance emulator of G with respect to T; all edge weights of H are non-negative integers bounded by n.*

Our theorem provides a practical framework for solving distance-related problems on planar graphs: Replace the input graph G (or proper subgraphs of the right choice) with the corresponding distance emulator H, and invoke whatever algorithm available on H rather than on G. One concrete example of this is the computation of all-pairs shortest paths among a subset T of k vertices. The algorithm by Cabello [19] can compute these $\Theta(k^2)$ distances in $\tilde{O}(n^{4/3} + n^{2/3}k^{4/3})$ time. Mozes and Sommer [54] improved the running time to $\tilde{O}(n^{2/3}k^{4/3})$ when $k = \Omega(n^{1/4})$. Our emulator immediately implies a further improvement of the running time to $\tilde{O}(n + n^{1/2}k^{3/2})$ using an extremely simple algorithm: Replace G by the emulator H and run Dijkstra's single-source shortest-paths algorithm on H from every vertex of T. This improves on previous results by a polynomial factor for essentially the entire range of k.

▶ **Corollary 2.** *Let G be an undirected unweighted planar graph, and let T be a subset of k terminals in G. One can compute the shortest path distances between all pairs of vertices in T in $O(n \log^4 n + n^{1/2}k^{3/2} \log^3 n \log\log n)$ time.*

Techniques. The main novel idea is a graphical encoding of unit-Monge matrices. Our emulator is obtained by plugging this graphical encoding into the encoding scheme of Abboud et al. [4]. Their encoding consists of two parts. First, they explicitly store a set of distances between some pairs of vertices of G (not necessarily of T). It is trivial to turn this part into an emulator by adding a single weighted edge between every such pair. Second, they implicitly store all-pairs distances in a set containing pairs of the form (X, Y) where X is a *prefix* of a cyclic walk around a single face and Y is a *subset* of vertices of a cyclic walk around a (possibly different) single face. For each such pair (X, Y), the X-to-Y distances are efficiently encoded using only $O((|X| + |Y|) \cdot \log n)$ bits (rather than the naïve $O(|X| \cdot |Y| \cdot \log n)$ bits). This encoding follows from the fact that the $X \times Y$ distance matrix M is a *triangular unit-Monge* matrix.

A matrix is **Monge** if for any i, j we have that

$$M[i+1, j] - M[i, j] \;\leq\; M[i+1, j+1] - M[i, j+1].$$

A Monge matrix is **unit-Monge** if for any i, j we also have that

$$M[i+1, j] - M[i, j] \;\in\; \{-1, 0, 1\}.$$

A (unit-)Monge matrix is **triangular** if the above conditions are only required to hold for $i \neq j$. In other words, when all four entries $M[i, j], M[i+1, j], M[i, j+1], M[i+1, j+1]$ belong to the upper triangle of M or to the lower triangle of M.

The Monge property has been heavily utilized in numerous algorithms for distance related problems in planar graphs [33, 20, 54, 43, 48, 17, 18, 52] in computational geometry [46, 36, 6, 7, 5, 47], and in pattern matching [56, 27, 41, 59]. However, prior to the work of Abboud et al. [4], the stronger *unit-Monge* property has only been exploited in pattern matching [59, 58].

The encoding of the $X \times Y$ unit-Monge matrix M is immediate: For any i, the sequence of differences $M[i+1, j] - M[i, j]$ is nondecreasing and contains only values from $\{-1, 0, 1\}$, so they can be encoded by storing the positions of the first 0 and the first 1. Storing these positions for every i takes $O(|X| \cdot \log n)$ bits. To encode M, we additionally store $M[0, j]$ for every j using $O(|Y| \cdot \log n)$ bits. This encoding, however, is clearly not an emulator. Our main technical contribution is in showing that it can be turned into an emulator with no asymptotic overhead. Namely, in Section 2 we prove the following lemma.

▶ **Lemma 3.** *Given an $n \times n$ unit-Monge or triangular unit-Monge matrix M, one can construct in $O(n \log n)$ time a directed edge-weighted graph (emulator) H with $O(n \log n)$ vertices and edges such that the distance in H between vertex i and vertex j is exactly $M[i, j]$.*

In Section 3 we describe how to plug the emulator of Lemma 3 into the encoding scheme of Abboud et al.[4]. In their paper, the size of the encoding was the only concern and the construction time was not analyzed. In our case, however, the construction time is crucial for the application in Corollary 2. We achieve an $O(n \log^4 n)$ time construction by making a small modification to their encoding, based on the technique of heavy path decomposition.

Another difference is that in their construction once the encoding is computed, single-source shortest-paths can be computed on the encoding itself using an algorithm by Fakcharoenphol and Rao [33]. More precisely (see [4, Section 5]), using a compressed representation of unit-Monge matrices, the total size of their encoding is $s = O(\sqrt{k \cdot n} \log^2 n)$ words, running the Fakcharoenphol and Rao algorithm requires accessing $O(s \log^2 s)$ elements of the unit-Monge matrices, and accessing each such element from the compressed representation requires $O(\log n / \log \log n)$ time (via a range dominance query, see Section 2)[5]. Overall, using our $O(n \log^4 n)$ construction together with the above single-source shortest-paths computations from each terminal would result in a time bound of $O(n \log^4 n + n^{1/2} k^{3/2} \log^5 n / \log \log n)$ in Corollary 2. Our approach on the other hand, runs Dijkstra's classical algorithm on the emulator H, leading to a faster (by a $(\log n / \log \log n)^2$ factor) time bound for Corollary 2. More precisely, our emulator H is of size $O(\sqrt{k \cdot n} \log^3 n)$, the edge-weights in H are encoded explicitly (i.e. there is no need for a range dominance query), and the edge-weights are integers in $[1 \mathinner{..} n]$ so Dijkstra's algorithm can use an $O(\log \log n)$ time heap (using such fast heap is not possible for Fakcharoenphol and Rao's algorithm). Running Dijkstra therefore takes $O(\sqrt{k \cdot n} \log^3 n \log \log n)$ time leading to the bound in Corollary 2.

2 Distance Emulators for Unit-Monge Matrices

2.1 The bipartite case

We next describe an $O(n \log n)$ space and time construction of a distance emulator for *square* $n \times n$ unit-Monge matrices. The construction can be trivially extended to *rectangular* $n \times m$ unit-Monge matrices (in $O(\max(n, m) \log(\max(n, m)))$ time and space) by duplicating the last row or column until the matrix becomes square.

We begin by establishing a relation between unit-Monge matrices and right substochastic binary matrices. A binary matrix P is **right substochastic** if every row of P contains at most one nonzero entry. The following lemma can be established similarly to Abboud et al. [4, Lemma 6.1].[6]

[5] The $O(\log n / \log \log n)$ factor was overlooked in [4]. It can be obtained by representing the unit-Monge matrix with a range dominance data structure, such as the one in [44].

[6] In Abboud et al. [4, Section 6] a unit-Monge matrix is defined to be one where every two adjacent elements differ by at most 1, whereas here we only assume this for elements that are adjacent vertically (that is, in the same column and between adjacent rows).

$$\begin{bmatrix} 5 & 3 & 3 & 3 & 3 & 2 & 1 \\ 4 & 3 & 3 & 4 & 4 & 3 & 2 \\ 3 & 3 & 3 & 4 & 5 & 4 & 3 \\ 2 & 2 & 2 & 3 & 4 & 4 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \Rightarrow$$

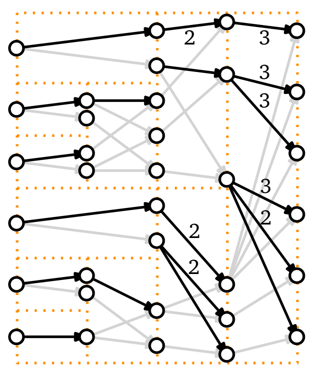


Figure 1 Unit-Monge distance matrix M, its corresponding row substochastic matrix P, and the emulator H for the matrix P. In this example the two vectors given by Lemma 4 are $U = [-3 -2 -1 0]$ and $V = [2\ 2\ 2\ 3\ 4\ 4\ 4]$. The leftmost vertices in H are row vertices $r[x]$, and the rightmost vertices in H are column vertices $c[y]$. All gray directed edges without labels have weight 0, and all black directed edges without labels have weight 1. For instance, the distance from $r[2]$ to $c[1]$ is equal to the number of 1s in P dominated by the $(2, 1)$-entry, which is 5.

▶ **Lemma 4** (Abboud et al. [4, Lemma 6.1]). *For any $n \times m$ unit-Monge matrix M there is a right substochastic $2(n-1) \times (m-1)$ binary matrix P such that for all x and y,*

$$M[x, y] \;=\; U[x] + V[y] + \sum_{\substack{i \geq 2x-1 \\ j \geq y}} P[i, j]$$

where U is a vector of length n and V is a vector of length m.

Our goal is to construct a small emulator (in terms of number of vertices and edges) for M. By Lemma 4, it suffices to construct a graph H satisfying

$$d_H[r[x], c[y]] \;=\; \sum_{\substack{i \geq x \\ j \geq y}} P[i, j],$$

where x and y range over the rows and columns of P, $r[x]$ is a vertex corresponding to row x of P, $c[y]$ is a vertex corresponding to column y of P, and the $r[x]$-to-$c[y]$ distance in H equals the number of 1s in P dominated by entry $P[x, y]$ (a 2-dimensional range dominance counting). We assume that P is given as input, which is represented as a vector specifying, for each row i of P, the index of the (at most) single one entry in that row. We refer to such a graph H as an **emulator** for the right substochastic matrix P. To convert an emulator for P into an emulator for M, add a new vertex $r_0[x]$ for each $1 \leq x < n$, and connect it with an edge of weight $U[x]$ to $r[2x-1]$. Similarly, for each $1 \leq y < m$, connect $c[y]$ to a new vertex $c_0[y]$ with an edge of weight $V[y]$. In addition, add a new vertex $c_0[m]$ and connect each $r_0[x]$ to $c_0[m]$ by an edge of weight $U[x] + V[m]$; and add a new vertex $r_0[n]$ and connect $r_0[n]$ to each $c_0[y]$ by an edge of weight $U[n] + V[y]$. By Lemma 4, the $r_0[x]$-to-$c_0[y]$ distance equals $M[x, y]$. We therefore focus for the remainder of this section on constructing an emulator for the right substochastic matrix P.

Recursive construction. We now describe how to construct an emulator for an $n \times \alpha n$ right substochastic matrix P recursively for arbitrary constant $\alpha \geq 0$.

When P has only a single row, we create a row vertex r for the single row and a column vertex $c[y]$ for each column y. Connect r to $c[y]$ with a directed edge of weight $\sum_{j \geq y} P[1, j]$.

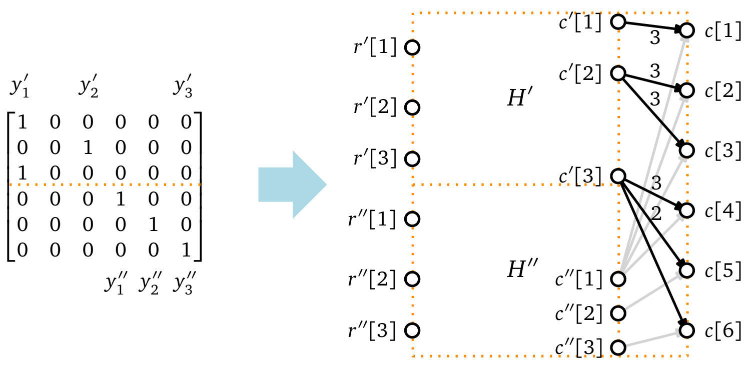


Figure 2 The top-level construction of the emulator. As an example, the edge from $c'[2]$ to $c[2]$ exists because $y'_{\to} = 2$ when $y = 2$, and the weight on the edge is equal to $\sum_{i>3,j\geq 2} P[i,j] = 3$.

Otherwise, assume P has more than one single row. Divide P into the top $\lfloor n/2 \rfloor \times \alpha n$ submatrix $\boldsymbol{P_\uparrow}$ and the bottom $\lceil n/2 \rceil \times \alpha n$ submatrix $\boldsymbol{P_\downarrow}$. Since $P_\uparrow$ is right substochastic, $P_\uparrow$ has at most $\lfloor n/2 \rfloor$ columns that are not entirely zero; similarly $P_\downarrow$ has at most $\lceil n/2 \rceil$ non-zero columns. Let $\boldsymbol{P'}$ and $\boldsymbol{P''}$ be the submatrices of $P_\uparrow$ and $P_\downarrow$ respectively, induced by their non-zero columns as well as their last column (if it's a zero vector); denote the number of columns of P' and P'' as n' and n'', respectively. Observe that both P' and P'' are still right substochastic matrices. Let $y'_1, \ldots, y'_{n'}$ be the indices in $P_\uparrow$ (and thus in P) corresponding to the columns of P', and let $y''_1, \ldots, y''_{n''}$ be the indices in $P_\downarrow$ corresponding to the columns of P''. Observe that for any row x, the sum

$$\sum_{\substack{i\geq x\\ j\geq y}} P_\uparrow[i,j]$$

is the same for all $y \in (y'_{\ell-1} \mathinner{..} y'_\ell]$ for any fixed ℓ.[7] This is because all the columns of $P_\uparrow$ in the range $(y'_{\ell-1} \mathinner{..} y'_\ell - 1]$ are all zero. A similar observation holds for the matrix $P_\downarrow$. For each y, we denote by $\boldsymbol{y'_\to}$ the (unique) smallest index $y'_\ell \in \{y'_1, \ldots, y'_{n'}\}$ no smaller than y, and denote by $\boldsymbol{y''_\to}$ the smallest index $y''_\ell \in \{y''_1, \ldots, y''_{n''}\}$ no smaller than y.

We recursively compute emulators H' and H'' for P' and P'' respectively. Denote the row and column vertices of H' as $r'[x]$ and $c'[y]$ where x and y range over the rows and columns of P', respectively; similarly denote the row and column vertices of H'' as $r''[x]$ and $c''[y]$ respectively. Take the row vertices of H' and H'' as the row vertices of H, respecting the indices of P. For each column y of P we add a column vertex $c[y]$ to H; connect vertex $c''[y''_\to]$ of H'' to $c[y]$ with an edge of weight zero; and connect vertex $c'[y'_\to]$ of H' to $c[y]$ with an edge of weight

$$\sum_{\substack{i>\lfloor n/2\rfloor\\ j\geq y}} P[i,j].$$

Thus, since for each pair of indices (x,y) of P one has

$$d_{H'}[r[x], c'[y]] = \sum_{\substack{x\leq i\leq \lfloor n/2\rfloor\\ j\geq y}} P_\uparrow[i,j] \quad\text{and}\quad d_{H''}[r[x], c''[y]] = \sum_{\substack{i\geq x\\ j\geq y}} P_\downarrow[i,j],$$

[7] For simplicity, one assumes $y'_0 = y''_0 = -\infty$.

we have that

$$d_H[r[x], c[y]] \;=\; \sum_{\substack{i \geq x \\ j \geq y}} P[i, j]$$

as desired. This completes the description of the construction of the emulator for square right substochastic matrices. Observe that the obtained emulator is a directed acyclic graph (dag) with a single path from each row vertex to each column vertex.

Size analysis. We next analyze the size of the emulator. Row vertices are created only at the base of the recursion and there are n of them in total. The number of edges and non-row vertices $N(n)$ satisfies the recurrence $N(n) \leq 2N(n/2) + O(\alpha n)$, so there are overall $O(n \log n)$ vertices and edges in the emulator when the number of columns is linear to the number of rows n.

Time analysis. As for the construction time, the only operation in the recursive procedure that does not clearly require constant time is the computation of the edge weights. However, it is not hard to see that computing all the edge weights of the form

$$w_y \;:=\; \sum_{\substack{i > \lfloor n/2 \rfloor \\ j \geq y}} P[i, j]$$

in a single recursive call can be done in linear time. This is because w_y is precisely the weight of the unique path in the emulator H'' from the first row vertex to the y'th column vertex. We can therefore simply maintain (with no asymptotic overhead), for every recursively built emulator, the distances from its first row vertex to all its column vertices.

Making weights non-negative. The weight of every edge in our emulator for right substochastic matrix P is non-negative, but the vectors U and V might contain negative entries; as a result the obtained emulator for M might contain negative edges even though every entry in M is non-negative. This can be fixed by a standard reweighting strategy using a *price function*. For the sake of computing the price function $\phi(\cdot)$, add a super-source s and connect s to each row vertex $r[x]$ of the emulator of M by an edge of weight zero. Compute the single-source shortest path tree rooted at s; since our emulator for P is a dag, this can be done in time linear in the size of the emulator. Take the shortest path distances $d(s, \cdot)$ as the price function $\phi(\cdot)$, and let the *reduced weight* $w_\phi(uv)$ of edge uv be $w(uv) + \phi(u) - \phi(v)$. Finally increase the weight of every incoming edge to $c[y]$ by the amount $\phi(y)$. Now all modified weights are non-negative, and all shortest path distances from $r[x]$ to $c[y]$ are preserved because $\phi(r[x]) = 0$ holds for all row vertices $r[x]$.

2.2 The non-bipartite case

Recall that the matrices that capture pairwise distances among consecutive vertices on the boundary of a single face in a planar graph are not (unit-)Monge. In Abboud et al. [4] this is handled by decomposing such a matrix recursively into (unit-)Monge submatrices, which incurs a logarithmic overhead. We show how to avoid this logarithmic overhead by constructing emulators for triangular (unit-)Monge matrices. Let $v_1, v_2, \ldots, v_n$ be vertices on the boundary of a single face. Let M be an $n \times n$ matrix such that $M[i, j]$ is the distance from v_i to v_j. For any quadruple (i, i', j, j') satisfying $i < i' < j < j'$, the shortest v_i-to-v_j path must intersect the shortest $v_{i'}$-to-$v_{j'}$ path. Therefore, $M[i,j] + M[i',j'] \geq M[i,j'] + M[i',j]$. Unfortunately, this does not necessarily hold for an arbitrary quadruple. We can, however, define two $n \times n$ unit-Monge matrices M_l and M_u, such that

$$\begin{aligned} M_l[i,j] &= M[i,j] && \text{for all } n \geq i > j \geq 1, \text{ and} \\ M_u[i,j] &= M[i,j] && \text{for all } 1 \leq i < j \leq n. \end{aligned}$$

The other elements of M_l and M_u can be (implicitly) filled in so that both matrices are unit-Monge. We would like to use an emulator for M_l and an emulator for M_u, but we need to eliminate paths from r_i to c_j for $i < j$ in M_l, and for $i > j$ in M_u. To this end we modify the construction of the emulator. For ease of presentation, we describe the construction for an $(n/2) \times n$ right substochastic matrix P in which we want the distance from c_i to r_j to be infinite for $i > j$. It is easy to modify the construction to work for $(n/2) \times \alpha n$ matrices, for $2n \times \alpha n$ matrices, or for the case where the infinite distances are for $i < j$. The elements of P are classified into two types: an element $P[i,j]$ of P is **artificial** if $i > j$ and **original** otherwise (that is, the i'th row of P begins with a prefix of $i-1$ artificial elements followed by a suffix of original elements). In the following recursive algorithm, the type of elements is always defined with respect to the full input matrix P (at the top level of the recursion).

We divide the $(n/2) \times n$ matrix P into two $(n/4) \times n$ matrices $P_\uparrow$ and $P_\downarrow$. Both $P_\uparrow$ and $P_\downarrow$ have possibly empty prefix of columns containing just artificial elements (category I), then an infix of columns containing both original and artificial elements (category II), and finally a suffix of columns containing just original elements (category III). Let P' be the submatrix of $P_\uparrow$ induced by all the columns of category II and by the columns of category III that are not entirely zero. Define submatrix P'' of $P_\downarrow$ similarly. Note that, by definition of artificial elements (with respect to the original top-level matrix), the number of columns of category II is bounded by the number of rows in $P_\uparrow$ and $P_\downarrow$. Also, by definition of right substochastic matrix, the number of columns of category III that are not entirely zero is also bounded by the number of rows of $P_\uparrow$ and $P_\downarrow$. Hence, both P' and P'' are $(n/4) \times (n/2)$ matrices for which we can recursively compute emulators H' and H'', respectively.

We construct the emulator H of P from the emulators H' and H'' as follows. For each column y of P we add a vertex $c[y]$ to H. If column y in $P_\uparrow$ consists of just artificial elements (category I), we do nothing. If column y in $P_\uparrow$ is of category II, then column y is also present in P', so it has a corresponding column vertex $c'[y]$ in H'. Connect $c'[y]$ to $c[y]$. If column y is of category III, connect vertex $c'[y'_\rightarrow]$ of H' to $c[y]$ with an edge of weight

$$\sum_{\substack{i > \lfloor n/4 \rfloor \\ j \geq y}} P[i,j].$$

Treat all the columns in $P_\downarrow$ similarly, except when column y is of category III, connect vertex $c''[y''_\rightarrow]$ of H'' to $c[y]$ with an edge of weight zero. (For the definition of $y'_\rightarrow$ and $y''_\rightarrow$ see the proof of the bipartite case.) An easy inductive proof shows that in this construction r_i is connected to c_j if and only if $i < j$.

The size of the emulator again satisfies the recurrence $N(n) \leq 2N(n/2) + O(n)$, and therefore is of $O(n \log n)$ overall. This concludes the proof of Lemma 3.

3 Distance Emulators for Planar Graphs

In this section we explain how to incorporate the unit-Monge emulators of Lemma 3 into the construction of planar graph emulators. The construction follows closely to the encoding scheme of Abboud et al. [4]. The crucial difference is that their unit-Monge matrices are represented by a compact non-graphical encoding, which we replace by the distance emulator of Lemma 3. There are additional differences that stem from the fact that Abboud et al. concentrated on the size of the encoding, and did not consider the construction time. To achieve efficient construction time we have to modify the construction slightly. These modifications are not directly related to the unit-Monge distance emulators. We will describe the construction algorithm without going into details, which were treated in Abboud et al. [4]. Instead, we attempt to give some intuition and describe the main components of the construction and their properties. In the full version of this paper we give a self-contained detailed description.

Compact representations of distances in planar graphs have been achieved in many previous works by decomposing the graph using small cycle separators and exploiting the Monge property (not the unit-Monge property) to compactly store the distances among the vertices of the separators. The main obstacle in exploiting the unit-Monge property using this approach is that small cycle separators do not necessarily exist in planar graphs with face of unbounded size. In weighted graphs this difficulty is overcome by triangulating the graph with edges with sufficiently large weights that do not affect distances. In unweighted graphs, however, this does not work.

Slices. To overcome the above difficulty, Abboud et al. showed that any unweighted planar graph contains a family of nested subgraphs (called **components**) with some desirable properties. This nested family can be represented by a tree $\mathcal{K}$, called the **component tree**, in which the ancestry relation corresponds to enclosure in G. Each component K in $\mathcal{K}$ is a subgraph of G that is enclosed by a simple cycle ∂K, called the **boundary cycle** of K. The root component is the entire graph G, and its boundary cycle is the infinite face of G. A **slice** $\boldsymbol{K^\circ}$ is defined as the subgraph of G induced by all faces of G that are enclosed by component K, and not enclosed by any descendant of K in $\mathcal{K}$. The boundary cycle ∂K is called the **external hole** or the **boundary** of K°. The boundary cycles of the children of K in $\mathcal{K}$ are called the **internal holes** of K°. Note that a slice may have many internal holes. Abboud et al. showed that, for any choice of $w \in [n]$, there exists a component tree $\mathcal{K}$ such that the total number of vertices in the boundaries of all the slices is $O(n/w)$ [4, Lemma 3.1]. We present this proof in the full version of this paper.

Regions. Constructing distance emulator for each slice separately is not good enough naïvely, as there could potentially be $\Omega(n/w)$ holes in a slice, and we cannot afford to store distances between each pair of holes, even if we use the unit-Monge property. To overcome this, each slice K° is further decomposed recursively into smaller subgraphs called **regions**. A *Jordan cycle separator* is a closed curve in the plane that intersects the embedding only at $O(w)$ vertices. A slice K° is recursively decomposed in a process described by a binary **decomposition tree** $\boldsymbol{\mathcal{R}_K}$; each node R of the tree $\mathcal{R}_K$ is a region of K°, with the root being the entire slice K°. There is a unique Jordan cycle separator C_R corresponding to each internal node R of $\mathcal{R}_K$. We abuse the terminology by referring to an internal hole of slice K° lying completely inside region R as a *hole* of R. We say that a Jordan curve C_R *crosses* a hole H of R if C_R intersects both inside and outside of ∂H. All Jordan cycle separators discussed here are mutually non-crossing (but may share subcurves) and each crosses a hole at most twice.

We next describe how the separators C_R are chosen. Let K be a component, and let K' be the child component of K in $\mathcal{K}$ with the largest number of vertices. We designate the hole $\partial K'$ of the slice K° as the **heavy** hole of K°.[8] For any region R of K°, let $\boldsymbol{R^\bullet}$ denote the union of R and all the subgraphs of G enclosed by the internal holes of R except for the heavy hole of K° (in other words, $R^\bullet$ is the region R with all its holes except the heavy hole "filled in"). For example, given a slice K°, one has $(K^\circ)^\bullet = K \setminus \mathit{int}(\partial H^*)$, where $\mathit{int}(\partial H^*)$ is the interior of the heavy hole H^* of K° (if any). Abboud et al. [4, Lemma 3.4] proved that for any region R one of the following must hold (and can be applied in time linear in the size of R):

- One can find a small Jordan cycle separator C_R that crosses no holes of R and is balanced with respect to the number of terminals in $R^\bullet$. That is, C_R encloses a constant fraction of the weight with respect to a weight function assigning each terminal in R unit weight and each face corresponding to a non-heavy hole of R weight equal to the number of terminals enclosed (in $R^\bullet$) by that hole. Such a separator is called a **good** separator.
- There exists a hole H such that one can *recursively* separate R with small Jordan cycle separators, each of which crosses H exactly twice, crosses no other hole, and is balanced with respect to the number of vertices of ∂H in the region R. It is further guaranteed that each resulting region R' along this recursion satisfies that $(R')^\bullet$ contains at most half the terminals in $R^\bullet$. The hole H is called a **disposable hole**, and the recursive process is called the **hole elimination** process. The hole elimination process terminates if a resulting region R' either contains a constant number of vertices of the disposable hole ∂H, or if $(R')^\bullet$ contains no terminals. Thus, the hole elimination process is represented by an internal subtree of $\mathcal{R}_K$ of height $O(\log n)$.

We begin with the region consisting of the entire slice K°. If a good separator is found, we use it to separate a region R into two subregions, each containing at most a constant fraction of the terminals in $R^\bullet$. If no good separator is found then we use the hole elimination process on a disposable hole H. We stop the decomposition as soon as a region contains at most one hole (other than the heavy hole) or at most one terminal. Thus, the overall height of $\mathcal{R}_K$ is $O(\log k \cdot \log n)$. We mentioned that it is guaranteed that each Jordan cycle separator used crosses a hole at most twice. This implies that, within each region R, the vertices of each hole ∂H of K° that belong to R can be decomposed into $O(\log k \cdot \log n)$ intervals called **$\boldsymbol{\partial H}$-intervals**; each Jordan cycle separator increases the number of such intervals in each subregion by at most one. We refer to those intervals that belong to the boundary cycle ∂K of K° or to the heavy hole ∂H^* of K° as **boundary intervals**.

3.1 The construction

We are now ready to describe the construction of the distance emulator. We will build the distance emulator of G from the leaves of the component tree $\mathcal{K}$ to the root (which is the whole graph G). For each component K in $\mathcal{K}$, we will associate an emulator that preserves the following distances.

- *terminal-to-terminal* distances in K between any two terminals in K.
- *terminal-to-boundary* distances in K between any terminal in K and vertices of boundary ∂K.

We emphasize that the distances being preserved are those in K, not K°. The distance emulator for G can be recovered from the emulator associates with the root of the component tree $\mathcal{K}$.

[8] This is the main difference compared with the encoding scheme of Abboud et al. [4]. We will see the benefit of this technicality in Section 3.3.

We now describe the construction of the emulator associated with component K, assuming all the emulators for the children of K in $\mathcal{K}$ have been constructed. The emulator of K is constructed by adding the following edges and unit-Monge emulators from Lemma 3 to the emulators obtained from the children of K.

1. Add unit-Monge emulators representing the distances in K° between all pairs of vertices on the *boundary* ∂K of K or on the *heavy hole* ∂H^* of K.
2. For each region R in the decomposition tree $\mathcal{R}_K$ of the slice K°, add the following edges and the unit-Monge emulators according to the type of R.
   - **(A)** If R is an internal node of $\mathcal{R}_K$ with separator C_R (either good or hole-eliminating):
     - **i.** Add an edge between each *terminal* in $R^\bullet$ and each vertex on the *separator* C_R. The weight of each edge is the distance in $R^\bullet$ between the two vertices.
     - **ii.** For each of the two subregions R_1 and R_2 of R, add a unit-Monge emulator representing the distance in $R_i^\bullet$ between the *separator* C_R and each *boundary interval* of R_i.
     - **iii.** If R corresponds to a cycle separator C_R eliminating a disposable hole ∂H:
       - **(a)** If R is the root of the subtree of $\mathcal{R}_K$ representing the hole-elimination process of a *hole* H, add a unit-Monge emulator representing the distances between ∂H and each *boundary interval* of R. The distances are in the subgraph obtained from $R^\bullet$ by removing the subgraph strictly enclosed by ∂H.
       - **(b)** For each of the two subregions R_1 and R_2 of R, add a unit-Monge emulator representing the distance in $R_i^\bullet$ between the *separator* C_R and each *∂H-interval* of R_i.
   - **(B)** If R is a leaf of $\mathcal{R}_K$ and all terminals in $R^\bullet$ are enclosed by a single non-heavy hole ∂H:
     - **i.** Add a unit-Monge emulator representing the distances between ∂H and each *boundary interval* of R. The distances are in the subgraph obtained from $R^\bullet$ by removing the subgraph strictly enclosed by ∂H.
   - **(C)** If R is a leaf of $\mathcal{R}_K$ that contains exactly one terminal t:
     - **i.** Add an edge between the only *terminal* t and each vertex on each *boundary interval* of R. The weight of each edge is the distance in $R^\bullet$ between the two vertices.

The proof of correctness is essentially the same as that in Abboud et al. [4, Lemma 3.6].

3.2 Space analysis

The $O(\sqrt{k \cdot n}\log^3 n)$ space analysis is essentially the same as in Abboud et al. [4]. The main difference in the algorithm is the introduction of the heavy hole as part of the boundary of a slice. This will allow an efficient construction time and does not affect the construction space because it increases the total size of the boundary by a constant factor.

Since a boundary cycle appears at most twice as a boundary of a slice (once as the external boundary and once as the heavy hole), the total size for all unit-Monge emulators for boundary-to-boundary distance (1) is $O((n/w)\log n)$.

We next bound the total space for terminal-to-separator distances (type 2(A)i) over all slices. Whenever a terminal stores its distance to a separator, the total number of terminals in its region $R^\bullet$ decreases by a constant factor within $O(\log n)$ levels of the decomposition tree (either immediately if the separator is a good separator, or within $O(\log n)$ levels of the hole elimination process), so this can happen $O(\log^2 n)$ times per terminal. Since each separator has size $O(w)$, the total space for representing distances of type 2(A)i is $O(kw\log^2 n)$.

The space required to store a single unit-Monge distance emulator representing distances between a separator C_R and a boundary interval b (type 2(A)ii) is $O(|C_R| + |b| \log |b|)$ (The rows of such an emulator correspond to the vertices of b, and the columns to vertices of C_R. Each row in the corresponding right sub-stochastic matrix has at most a single non-zero entry). We note the following facts: (i) The size of any separator C_R is $O(w)$. (ii) The depth of the recursion within each slice is $O(\log^2 n)$. (iii) There are at most k regions at each level of the recursion. (iv) The total number of boundary intervals in all regions at a specific recursive level is within a constant factor from the number of regions at that level, i.e., $O(k)$. Thus, the total size of all boundary intervals at a fixed level of the recursion is at most $O(k)$ plus the size of the boundary of the slice (boundary intervals are internally disjoint, and there are $O(k)$ intervals in each region). (v) The total boundary size of all slices combined is $O(n/w)$. By the above facts, the total space for type 2(A)ii emulators is $O((kw + (k + n/w) \log n) \log^2 n) = O((kw + n/w) \log^3 n)$.

Hole-to-boundary distances (types 2(A)iiia and 2(B)i) are stored for at most one hole in each region along the recursion, and require $O(|\partial H| + |b| \log |b|)$ space. By the same arguments as above, and since the total size of hole boundaries ∂H is bounded by the total size of slice boundaries, the total space for types 2(A)iiia and 2(B)i emulators is $O((k + n/w) \log^3 n)$.

The space for separator-to-hole emulators (type 2(A)iiib) is bounded by the separator-to-boundary distances (type 2(A)ii).

To bound the number of terminal-to-boundary edges (type 2(C)i) note that each terminal stores distances to boundary intervals that are internally disjoint for distinct terminals. Since the number of boundary terminals in a region is $O(\log n)$, the total space for type 2(C)i distances is $O(k \log n + n/w)$.

The overall space required is thus $O((kw + n/w) \log^3 n)$; setting $w = \sqrt{k \cdot n}$ gives the bound $O(\sqrt{k \cdot n} \log^3 n)$.

3.3 Time analysis

Recall that Lemma 3 assumes that the input is an $n \times n$ unit-Monge matrix represented in $O(n \log n)$ bits as two arrays U and V, as well as an array P, specifying, for each row i of the right substochastic $2(n-1) \times (n-1)$ matrix P, the index of the (at most) single one entry in that row (see Lemma 4). Lemma 3 will be invoked on various $x \times x$ unit-Monge matrices that represent distances among a set of x vertices on the boundary of a single face of a planar graph G with n vertices (or sometimes, on two sets of vertices on two distinct faces).

▶ **Lemma 5.** *Let G be a planar graph with n nodes. Let f be a face in G. One can compute the desired representation of the unit-Monge matrix M representing distances among the vertices of f in $O(n)$ time.*

Proof. We tweak the linear-time multiple source shortest paths (MSSP) algorithm of Eisentat and Klein [29]. Along its execution, the MSSP algorithm maintains the shortest path trees rooted at each vertex of a single distinguished face f, one after the other. This is done by transforming the shortest path tree rooted at some vertex v_i to the shortest path tree rooted at the next vertex v_{i+1} in cyclic order along the distinguished face f. Let $d_i(u)$ denote the distance from the current root v_i to v. The slack of an arc uw with respect to root v_i is defined as $slack_i(uw) := c(uw) + d_i(u) - d_i(w)$. Thus, for all arcs in the shortest path tree the slack is zero, and for all arcs not in the shortest path tree the slack is non-negative. Eisenstat and Klein show that throughout the entire execution of the algorithm there are $O(n)$ changes to the slacks of edges. Their algorithm explicitly maintains and updates the slacks of all edges as the root moves along the distinguished face. By definition of the matrix P,

$$P[i,j] \;=\; M[i+1,j+1]+M[i,j]-M[i+1,j]-M[i,j+1].$$

Consider an edge v_jv_{j+1} along the distinguished face. Its slack with respect to root v_i is $1+M[i,j]-M[i,j+1]$. Its slack with respect to root v_{i+1} is $1+M[i+1,j]-M[i+1,j+1]$. Therefore, $P[i,j] = \mathit{slack}_i(v_jv_{j+1}) - \mathit{slack}_{i+1}(v_jv_{j+1})$. It follows that one can keep track of the nonzero entries of P by keeping track of the edges of the distinguished face whose slack changes. The entries of the arrays U and V can be obtained in total $O(n)$ time since they only depend on the distances from a single vertex of f. ◀

We can now analyze the construction time. The boundary-to-boundary distances (type 1) in a single slice are computed by a constant number of invocations of MSSP (Lemma 5) on $K^\bullet$. We then compute the emulator for each unit-Monge matrix on x vertices in $O(x\log x)$ time using Lemma 3. For the total MSSP time, we need to bound the total size of the regions $R^\bullet$ on which we invoke Lemma 5. This is where we use the fact that the heavy hole of a slice K° contains at least half the vertices of K. Since $K^\bullet$ does not include the heavy hole of K, the standard heavy path argument implies that the number of slices in which a vertex v participates in invocations of the MSSP algorithm is at most 1 plus $O(\log n)$ (the number of non-heavy holes v belongs to). Thus, each vertex v participates in invocations of MSSP for $O(\log n)$ slices. In each invocation of MSSP each vertex v is charged $O(\log n)$ time, so the total time for all MSSP invocations is $O(n\log^2 n)$. To bound the total time of the invocations of Lemma 3, note that each boundary cycle appears at most twice as a boundary of a slice (once as the external boundary and once as the heavy hole). Hence, the total time for all invocations of Lemma 3 is $O((n/w)\log n)$. Overall we get that the total time to compute all the emulators of type 1 is $O(n\log^2 n + (n/w)\log n) = O(n\log^2 n)$.

The terminal-to-separator distances (type 2(A)i) are computed by running MSSP twice, once for the interior of the separator C_R in $R^\bullet$ and once for the exterior of C_R in $R^\bullet$. This takes $O(|R^\bullet|\log|R^\bullet|)$ time. Then, we can report the distance, within each of these two subgraphs of $R^\bullet$, from each vertex of C_R to any other vertex in $O(\log n)$ time. Next, for each terminal t in $R^\bullet$, we compute the distance from t to all w vertices of C_R by running FR-Dijkstra [33] on the graph consisting of the edges between t and the vertices of C_R as well as the two complete graphs representing the distances among the vertices of C_R in the interior and exterior. This takes $O(w\log^2 w\log n)$ time. The $O(w\log^2 w)$ term is the running time of FR-Dijkstra and the additional $O(\log n)$ factor is because each distance accesses by FR-Dijkstra is retrieved on-the-fly from the MSSP data structure.

We analyze separately the total cost of all the MSSP computations and the total cost of computing the edge weights. For the latter, we have argued in the analysis of the emulator's size that each terminal participates in the computation of edges of type 2(A)i in $O(\log^2 n)$ regions along the entire construction. Therefore, the total time to compute all such edges is $O(kw\log^2 w\log^3 n)$. For the total MSSP time, we need to bound the total size of the regions $R^\bullet$ on which we run MSSP. This is done in a similar manner to the bound on the time for MSSP invocations for type 1 above, using the standard heavy path argument. We argued that the number of slices in which a vertex v participates in invocations of the MSSP algorithm is at most 1 plus $O(\log n)$ (the number of non-heavy holes v belongs to). Thus, each vertex v participates in invocations of MSSP for $O(\log n)$ slices. Since the depth of the decomposition tree of a slice is $O(\log^2 n)$, each vertex participates in at most $O(\log^3 n)$ invocations of MSSP in the computation of edges of type 2(A)i. Each vertex is charged $O(\log n)$ time in each invocation of MSSP it participates in, so the total time for all invocations of MSSP in the process of computing edges of type 2(A)i is $O(n\log^4 n)$.

To bound the time required to compute the emulators of type 2(A)ii (separator-to-boundary distances), note that the appropriate unit-Monge matrices can be computed by MSSP invocations within the same time bounds analyzed for the MSSP invocations for terminal-to-separator edges above. Hence, we only need to account for the additional time spent on constructing the emulators by invocations of Lemma 3, which is linear in the total size of these emulators, which we have already shown above to be $O((kw + n/w)\log^3 n)$.

The analysis for the time required for the hole-to-boundary emulators (types 2(A)iiia and 2(B)i) uses the same arguments as the analysis for type 2(A)ii. Again, the MSSP time is $O(n \log^4 n)$, and the time for invocations of Lemma 3 is linear in the size of these emulators, which is $O((k + n/w)\log^3 n)$.

The time to compute the separator-to-hole emulators (type 2(A)iiib) is bounded by the time to compute the separator-to-boundary emulators (type 2(A)ii).

To compute the terminal-to-boundary edges we again need to invoke MSSP once on $R^\bullet$ for each leaf region R, and then query the distance in $O(\log n)$ time per distance. The time for MSSP is dominated by the MSSP time accounted for in type 2(A)ii emulators, and since we have shown that the total number of such edges is $O(k + n/w)$, the total time to compute them is $O((k + n/w)\log n)$.

The overall construction time is therefore dominated by the $O(n \log^4 n)$ term. Combined with the space analysis of Section 3.2, we establish Theorem 1.

References

1 Amir Abboud and Greg Bodwin. Error amplification for pairwise spanner lower bounds. In *27th SODA*, pages 841–854, 2016.

2 Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. *J. ACM*, 64(4):28:1–28:20, 2017.

3 Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. In *28th SODA*, pages 568–576, 2017.

4 Amir Abboud, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Near-optimal compression for the planar graph metric. In *29th SODA*, pages 530–549, 2018.

5 Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2(1):195–208, 1987.

6 Alok Aggarwal and James K. Park. Notes on searching in multidimensional monotone arrays (preliminary version). In *29th FOCS*, pages 497–512, 1988.

7 Alok Aggarwal and Subhash Suri. Fast algorithms for computing the largest empty rectangle. In *3rd SoCG*, pages 278–290, 1987.

8 Noga Alon. Testing subgraphs in large graphs. *Random Struct. Algorithms*, 21(3-4):359–370, 2002.

9 Stephen Alstrup, Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Ely Porat. Sublinear Distance Labeling. In *24th ESA*, pages 5:1–5:15, 2016.

10 Stephen Alstrup, Cyril Gavoille, Esben Bistrup Halvorsen, and Holger Petersen. Simpler, faster and shorter labels for distances in graphs. In *27th SODA*, pages 338–350, 2016.

11 Amitabh Basu and Anupam Gupta. Steiner point removal in graph metrics. *Unpublished Manuscript, available from http://www.math.ucdavis.edu/~abasu/papers/SPR.pdf*, 2008.

12 Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. New constructions of (α, β)-spanners and purely additive spanners. In *16th SODA*, pages 672–681, 2005.

13 Guy E. Blelloch and Arash Farzan. Succinct representations of separable graphs. In *21st CPM*, pages 138–150, 2010.

14 Greg Bodwin. Linear size distance preservers. In *28th SODA*, pages 600–615, 2017.

15 Greg Bodwin and Virginia Vassilevska Williams. Better distance preservers and additive spanners. In *27th SODA*, pages 855–872, 2016.

16 Béla Bollobás, Don Coppersmith, and Michael Elkin. Sparse distance preservers and additive spanners. *SIAM Journal on Discrete Mathematics*, 19(4):1029–1055, 2005.

17 Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. In *52nd FOCS*, pages 170–179, 2011.

18 Glencora Borradaile, Piotr Sankowski, and Christian Wulff-Nilsen. Min st-cut oracle for planar graphs with near-linear preprocessing time. In *51st FOCS*, pages 601–610, 2010.

19 Sergio Cabello. Many distances in planar graphs. *Algorithmica*, 62(1-2):361–381, 2012.

20 Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. In *28th SODA*, pages 2143–2152, 2017.

21 Hubert T.-H. Chan, Donglin Xia, Goran Konjevod, and Andréa W. Richa. A tight lower bound for the steiner point removal problem on trees. In *9th APPROX*, pages 70–81, 2006.

22 Hsien-Chih Chang and Hsueh-I Lu. Computing the girth of a planar graph in linear time. *SIAM Journal on Computing*, 42(3):1077–1094, 2013.

23 Yun Kuen Cheung, Gramoz Goranci, and Monika Henzinger. Graph minors for preserving terminal distances approximately - lower and upper bounds. In *43rd ICALP*, pages 131:1–131:14, 2016.

24 Yi-Ting Chiang, Ching-Chi Lin, and Hsueh-I Lu. Orderly spanning trees with applications to graph encoding and graph drawing. In *Proc. 12th Symp. Discrete Algorithms*, pages 506–515, 2001.

25 Eden Chlamtác, Michael Dinitz, Guy Kortsarz, and Bundit Laekhanukit. Approximating spanners and directed steiner forest: Upper and lower bounds. In *28th SODA*, pages 534–553, 2017.

26 Don Coppersmith and Michael Elkin. Sparse sourcewise and pairwise distance preservers. *SIAM Journal on Discrete Mathematics*, 20(2):463–501, 2006.

27 Maxime Crochemore, Gad. M. Landau, and Michal Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM Journal on Computing*, 32:1654–1673, 2003.

28 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.

29 David Eisenstat and Philip N Klein. Linear-time algorithms for max flow and multiple-source shortest paths in unit-weight planar graphs. In *45th STOC*, pages 735–744, 2013.

30 Michael Elkin, Yuval Emek, Daniel A Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM Journal on Computing*, 38(2):608–628, 2008.

31 Michael Elkin and Seth Pettie. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. *ACM Trans. Algorithms*, 12(4):50:1–50:31, 2016.

32 Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Racke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. *SIAM Journal on Computing*, 43(4):1239–1262, 2014.

33 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, 2006.

34 Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. *Journal of Algorithms*, 53(1):85–112, 2004.

35 Paweł Gawrychowski, Adrian Kosowski, and Przemysław Uznański. Sublinear-space distance labeling using hubs. In *30th DISC*, pages 230–242, 2016.

36 Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Submatrix maximum queries in Monge matrices are equivalent to predecessor search. In *42nd ICALP*, pages 580–592, 2015.

37 Gramoz Goranci, Monika Henzinger, and Pan Peng. Improved guarantees for vertex sparsification in planar graphs. In *25th ESA*, pages 44:1–44:14, 2017.

38 Gramoz Goranci and Harald Räcke. Vertex sparsification in trees. In *14th WAOA*, pages 103–115, 2016.

39 Ronald L Graham and Henry O Pollak. On embedding graphs in squashed cubes. In *Graph theory and applications*, volume 303, pages 99–110. Springer, 1972.

40 Anupam Gupta. Steiner points in tree metrics don't (really) help. In *12th SODA*, pages 220–227, 2001.

41 Danny Hermelin, Gad M. Landau, Shir Landau, and Oren Weimann. A unified algorithm for accelerating edit-distance via text-compression. *Algorithmica*, 65:339–353, 2013.

42 Shang-En Huang and Seth Pettie. Lower bounds on sparse spanners, emulators, and diameter-reducing shortcuts. In *16th SWAT*, pages 26:1–26:12, 2018.

43 Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *43rd STOC*, pages 313–322, 2011.

44 Joseph JáJá, Christian Worm Mortensen, and Qingmin Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *15th ISAAC*, pages 558–568, 2004.

45 Sampath Kannan, Moni Naor, and Steven Rudich. Implicat representation of graphs. *SIAM Journal on Discrete Mathematics*, 5(4):596–603, 1992.

46 Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. Submatrix maximum queries in Monge matrices and Monge partial matrices, and their applications. In *23rd SODA*, pages 338–355, 2012.

47 M. M. Klawe and D J. Kleitman. An almost linear time algorithm for generalized matrix searching. *SIAM Journal Discrete Math.*, 3(1):81–97, 1990.

48 Philip Klein, Shay Mozes, and Oren Weimann. Shortest paths in directed planar graphs with negative lengths: a linear-space $O(n \lg^2 n)$-time algorithm. *ACM Transactions on Algorithms*, 6(2):2–13, 2010.

49 Lukasz Kowalik and Maciej Kurowski. Short path queries in planar graphs in constant time. In *35th STOC*, pages 143–148, 2003.

50 Robert Krauthgamer, Huy L Nguyen, and Tamar Zondiner. Preserving terminal distances using minors. *SIAM Journal on Discrete Mathematics*, 28(1):127–141, 2014.

51 Robert Krauthgamer and Inbal Rika. Refined vertex sparsifiers of planar graphs. Preprint, July 2017.

52 Jakub Lacki, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Single source - all sinks max flows in planar digraphs. In *53rd FOCS*, pages 599–608, 2012.

53 J. W. Moon. On minimal n-universal graphs. *Glasgow Mathematical Journal*, 7(1):32–33, 1965.

54 Shay Mozes and Christian Sommer. Exact distance oracles for planar graphs. In *23rd SODA*, pages 209–222, 2012.

55 J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *Proc. 38th Annual Symposium on Foundations of Computer Science*, pages 118–126, 1997.

56 Jeanette P. Schmidt. All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings. *SIAM Journal on Computing*, 27(4):972–992, 1998.

57 Mikkel Thorup and Uri Zwick. Spanners and emulators with sublinear distance errors. In *17th SODA*, pages 802–809, 2006.

58 Alexander Tiskin. Semi-local string comparison: algorithmic techniques and applications. *Arxiv 0707.3619*, 2007.

59 Alexander Tiskin. Fast distance multiplication of unit-Monge matrices. In *21st SODA*, pages 1287–1296, 2010.

60 György Turán. On the succinct representation of graphs. *Discrete Applied Mathematics*, 8(3):289–294, 1984.

61 Oren Weimann and Raphael Yuster. Computing the girth of a planar graph in $O(n \log n)$ time. *SIAM J. Discrete Math.*, 24(2):609–616, 2010.

62 Peter M Winkler. Proof of the squashed cube conjecture. *Combinatorica*, 3(1):135–139, 1983.

63 David P Woodruff. Lower bounds for additive spanners, emulators, and more. In *47th FOCS*, pages 389–398, 2006.

64 Christian Wulff-Nilsen. Wiener index and diameter of a planar graph in subquadratic time. In *25th EuroCG*, pages 25–28, 2009.

65 Christian Wulff-Nilsen. Constant time distance queries in planar unweighted graphs with subquadratic preprocessing time. *Computational Geometry*, 46(7):831–838, 2013.

Approximation Schemes for Geometric Coverage Problems

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

Minati De[1]
Department of Mathematics, Indian Institute of Technology Delhi, India
minati@maths.iitd.ac.in
https://orcid.org/0000-0002-1859-8800

Alexander Ravsky
Pidstryhach Institute for Applied Problems of Mechanics and Mathematics,
National Academy of Science of Ukraine, Lviv, Ukraine
alexander.ravsky@uni-wuerzburg.de

Joachim Spoerhase[2]
Lehrstuhl für Informatik I, Universität Würzburg, Germany; and
Institute of Computer Science, University of Wrocław, Poland
joachim.spoerhase@uni-wuerzburg
https://orcid.org/0000-0002-2601-6452

Abstract

In their seminal work, Mustafa and Ray [30] showed that a wide class of geometric *set cover (SC)* problems admit a PTAS via local search – this is one of the most general approaches known for such problems. Their result applies if a naturally defined "exchange graph" for two feasible solutions is planar and is based on subdividing this graph via a planar separator theorem due to Frederickson [17]. Obtaining similar results for the related *maximum coverage problem (MC)* seems non-trivial due to the hard cardinality constraint. In fact, while Badanidiyuru, Kleinberg, and Lee [4] have shown (via a different analysis) that local search yields a PTAS for two-dimensional real halfspaces, they only conjectured that the same holds true for dimension three. Interestingly, at this point it was already known that local search provides a PTAS for the corresponding set cover case and this followed directly from the approach of Mustafa and Ray.

In this work we provide a way to address the above-mentioned issue. First, we propose a *color-balanced* version of the planar separator theorem. The resulting subdivision approximates locally in each part the global distribution of the colors. Second, we show how this roughly balanced subdivision can be employed in a more careful analysis to strictly obey the hard cardinality constraint. More specifically, we obtain a PTAS for any "planarizable" instance of MC and thus essentially for all cases where the corresponding SC instance can be tackled via the approach of Mustafa and Ray. As a corollary, we confirm the conjecture of Badanidiyuru, Kleinberg, and Lee [4] regarding real halfspaces in dimension three. We feel that our ideas could also be helpful in other geometric settings involving a cardinality constraint.

2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases balanced separators, maximum coverage, local search, approximation scheme, geometric approximation algorithms

[1] Partially supported by DST-INSPIRE Faculty Grant (DST-IFA-14-ENG-75)
[2] Partially supported by Polish National Science Centre grant 2015/18/E/ST6/00456

© Steven Chaplick, Minati De, Alexander Ravsky, and Joachim Spoerhase;
licensed under Creative Commons License CC-BY

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.17

Related Version A full version is available at `https://arxiv.org/abs/1607.06665`.

1 Introduction

The Maximum Coverage (MC) problem is one of the classic combinatorial optimization problems which is well studied due to its wealth of applications. Let U be a set of ground elements, $\mathcal{F} \subseteq 2^U$ be a family of subsets of U and k be a positive integer. The Maximum Coverage (MC) problem asks for a k-subset $\mathcal{F}'$ of $\mathcal{F}$ such that the number $|\bigcup \mathcal{F}'|$ of ground elements covered by $\mathcal{F}'$ is maximized.

Many real life problems arising from banking [13], social networks, transportation network [28], databases [22], information retrieval, sensor placement, security (and others) can be framed as an instance of MC problem. For example, the following are easily seen as MC problems: placing k sensors to maximize the number of covered customers, finding a set of k documents satisfying the information needs of as many users as possible [4], and placing k security personnel in a terrain to maximize the number of secured regions.

From the result of Cornuéjols [13], it is well known that the greedy algorithm is a $1 - 1/e$ approximation algorithm for the MC problem. Due to wide applicability of the problem, whether one can achieve an approximation factor better than $1 - 1/e$ was subject of research for a long period of time. From the result of Feige [16], it is known that if there exists a polynomial-time algorithm that approximates maximum coverage within a ratio of $1 - 1/e + \epsilon$ for some $\epsilon > 0$ then P = NP. Better results can, however, be obtained for special cases of MC. For example, Ageev and Sviridenko [1] show in their seminal work that their pipage rounding approach gives a factor $1 - (1 - 1/r)^r$ for instances of MC where every element occurs in at most r sets. For constant r this is a strict improvement on $1 - 1/e$ but this bound is approached if r is unbounded. For example, pipage rounding gives a 3/4-approximation algorithm for Maximum Vertex Cover (MVC), which asks for a k-subset of nodes of a given graph that maximizes the number of edges incident on at least one of the selected nodes. Petrank [31] showed that this special case of MC is APX-hard.

In this paper, we study the approximability of MC in *geometric* settings where elements and sets are represented by geometric objects. Such problems have been considered before and have applications, for example, in information retrieval [4] and in wireless networks [15].

MC is related to the Set Cover problem (SC). For a given set U of ground elements and a family $\mathcal{F} \subseteq 2^U$ of subsets of U, this problem asks for a minimum cardinality subset of $\mathcal{F}$ which covers all the ground elements of U. This problem plays a central role in combinatorial optimization and in particular in the study of approximation algorithms. The best known approximation algorithm has a ratio of $\ln n$, which is essentially the best possible [16] under a plausible complexity-theoretic assumption. A lot of work has been devoted to beating the logarithmic barrier in the context of geometric set cover problems [6, 33, 8, 29]. Mustafa and Ray [30] introduced a powerful tool which can be used to show that a *local search* approach provides a PTAS for various geometric SC problems. Their result applies if a naturally defined *exchange graph* (whose nodes are the sets in two feasible solutions) is planar and is based on subdividing this graph via a planar separator theorem due to Frederickson [17]. In the same paper [30], they applied this approach to provide a PTAS for the SC problem when the family $\mathcal{F}$ consists of either a set of halfspaces in $\mathbb{R}^3$, or a set of disks in $\mathbb{R}^2$. Concurrently with Mustafa and Ray [30], Chan and Har-Peled [9] also introduced the exchange graph concept, but for the independent set problem. Subsequently, many results have been obtained using this technique for problems in geometric settings [10, 14, 19, 26]. Some of these works extend to cases where the exchange graph is not planar but admits a small-size separator [3, 7, 20, 21].

Beyond the context of SC, local search has also turned out to be a very powerful tool for other geometric problems but the analysis of such algorithms is usually non-trivial and highly tailored to the specific setting. Examples of such problems are Euclidean TSP, Euclidean Steiner tree, facility location, k-median [12]. In some recent breakthroughs, PTASs for the k-means problem in finite Euclidean dimension (and more general cases) via local search have been published [11, 18].

In this paper, we study the effectiveness of local search for geometric MC problems. In the general case, b-swap local search is known to yield a tight approximation ratio of $1/2$ [24]. However, for special cases such as geometric MC problems local search is a promising candidate for beating the barrier $1 - 1/e$. It seems, however, non-trivial to obtain such results using the technique of Mustafa and Ray [30]. In their analysis, each part of the subdivided planar exchange graph (see above) corresponds to a feasible candidate swap that replaces some sets of the local optimum with some sets of the global optimum and it is ensured that every element stays covered due to the construction of the exchange graph. It is moreover argued that if the global optimum is sufficiently smaller than the local optimum then one of the considered candidate swaps would actually reduce the size of the solution.

It is possible to construct the same exchange graphs also for the case of MC. However, the hard cardinality constraint given by input parameter k poses an obstacle. In particular, when considering a swap corresponding to a part of the subdivision, this swap might be infeasible as it may contain (substantially) more sets from the global optimum than from the local optimum. Another issue is that MC has a different objective function than SC. Namely, the goal is to maximize the number of covered elements rather than minimizing the number of used sets. Finally, while for SC all elements are covered by both solutions, in MC we additionally have elements that are covered by none or only one of the two solutions requiring a more detailed distinction of several types of elements.

In fact, subsequent to the work of Mustafa and Ray on SC [30], Badanidiyuru, Kleinberg, and Lee [4] studied geometric MC. They obtained fixed-parameter approximation schemes for MC instances for the very general case where the family $\mathcal{F}$ consists of objects with bounded VC dimension, but the running times are exponential in the cardinality bound k. They further provided APX-hardness for each of the following cases: set systems of VC-dimension 2, halfspaces in $\mathbb{R}^4$, and axis-parallel rectangles in $\mathbb{R}^2$. Interestingly, while they have shown that for MC instances where $\mathcal{F}$ consists of halfspaces in $\mathbb{R}^2$ local search can be used to provide a PTAS, they only conjecture that local search provides a PTAS when $\mathcal{F}$ consists of halfspaces in $\mathbb{R}^3$. This underlines the observation that it seems non-trivial to apply the approach of Mustafa and Ray to geometric MC problems as at that point a PTAS for halfspaces in $\mathbb{R}^3$ for SC was already known via the approach of Mustafa and Ray.

The difficulty of analyzing local search under the presence of a cardinality constraint is also known in other settings. For example, one of the main technical contributions of the recent breakthrough for the Euclidean k-means problem [11, 18] is that the authors are able to handle the hard cardinality constraint by the concept of so-called isolated pairs [11]. Prior to these works approximation schemes have only been known for bicriteria variants where the cardinality constraint may be violated or where there is no constraint but – analogously to SC – the cardinality contributes to the objective function [5].

1.1 Our Contribution

We show a way to cope with the above-mentioned issue with a cardinality constraint. We are able to achieve a PTAS for many geometric MC problems using the b-swap local search approach given in Algorithm 1. At a high level we follow the framework of Mustafa and Ray defining a planar (or more generally *f-separable* as formalized below) exchange graph and subdividing it into a number of small parts each of them corresponding to a candidate swap. As each part may be (substantially) imbalanced in terms of the number of sets of the global optimum and local optimum, respectively, a natural idea is to swap-in only a sufficiently small subset of the globally optimal sets. This idea alone is, however, not sufficient. Consider, for example, the case where each part contains either only sets from the local or only sets from the global optimum making it impossible to retrieve any feasible swap from considering the single parts. To overcome this difficulty, in Section 2, we prove in a first step a *color-balanced* version (Theorem 7) of the planar division theorem (Theorem 4 [17]). In this theorem, the input is a planar (or more generally *f-separable*) graph whose nodes are two-colored arbitrarily. The distinctions of our division theorem from the prior work, are that our division theorem guarantees that all parts have roughly the same size (rather than simply an upper limit on their size) and that the two colors are represented in each part in roughly the same ratio as in the whole graph. This balancing property allows us to address the issue of the above-mentioned infeasible swaps. In a second step, described in Section 3, we employ this roughly color-balanced subdivision to establish a set of perfectly balanced candidate swaps. We prove by a careful analysis (which turns out more intricate than for the SC case) that local search also yields a PTAS for the wide class *f-separable* MC problems (see Theorem 2). As a direct consequence, we obtain PTASs for essentially all cases of geometric MC problems (see (Theorem 3)) where the corresponding SC problem can be tackled via the approach of Mustafa and Ray. For example, this confirms the conjecture of Badanidiyuru, Kleinberg, and Lee [4] regarding halfspaces in $\mathbb{R}^3$. We also obtain PTASs for MAXIMUM DOMINATING SET and MAXIMUM VERTEX COVER on f-separable and minor-closed graph classes (as formalized below) which, to the best of our knowledge, were not known before. We feel that our approach has the potential for further applications in similar cardinality constrained settings.

In the remainder of this section we formalize our main theorem (providing the needed definitions) and then present a set of applications of this theorem.

Definitions and the main theorem. For a number n, $[n]$ denotes the set $\{1,\ldots,n\}$. For a graph G, a subset S of $V(G)$ is an *α-balanced separator* when its removal breaks G into two collections of connected components such that each collection contains at most an α fraction of $V(G)$ where $\alpha \in [\frac{1}{2},1)$ and α is a constant. The size of a separator S is simply the number of vertices it contains. Let f be a non-decreasing (strictly) sublinear function, that is, $f(x) = O(x^{1-\delta})$ for some $\delta > 0$. A class of graphs that is closed under taking subgraphs is said to be *f-separable* if there is an $\alpha \in [\frac{1}{2},1)$ such that for any $n > 2$, an n-vertex graph in the class has an α-balanced separator whose size is at most $f(n)$. In what follows, whenever we discuss an f-separable graph classes we implicitly assume that the function f is non-decreasing and has the form $f(x) = O(x^{1-\delta})$ for some $\delta > 0$ – this is what we mean by *strictly* sublinear. Note that, by the Lipton-Tarjan separator theorem [27], planar graphs are a subclass of the $\sqrt{n}$-separable graphs. More generally, Alon, Seymour, and Thomas [2] have shown that every graph class characterized by a finite set of forbidden minors is also a subclass of the $(c \cdot \sqrt{n})$-separable graphs (here, the constant c depends on the size of the largest forbidden minor). In particular, from the graph minors theorem [32], every non-trivial

Algorithm 1: b-swap local search on an MC instance with ground set U, set family $\mathcal{S}$, and parameter k.

```
$b$-LocalSearch$(U, \mathcal{S}, k)$
$\mathcal{F} \leftarrow$ initialize with a greedy solution
while $\exists\ \mathcal{F}' \subseteq \mathcal{F}$, $\exists \mathcal{F}^* \subset \mathcal{S}$, such that $b = |\mathcal{F}^*| = |\mathcal{F}'|$ and
 $|\bigcup \mathcal{F}| < |\bigcup((\mathcal{F} \setminus \mathcal{F}') \cup \mathcal{F}^*)|$ do
    perform the swap, i.e., $\mathcal{F} \leftarrow (\mathcal{F} \setminus \mathcal{F}') \cup \mathcal{F}^*$
return $\mathcal{F}$
```

minor-closed graph class is a subclass of the $(c \cdot \sqrt{n})$-separable graphs (for some constant c). With this notion in mind, we define the concept of f-separable MC instances, then state our main theorem (the proof is given in Section 3).

▶ **Definition 1.** A class $\mathcal{C}$ of instances of MC is *f-separable* (or in particular *planarizable*) if for any two disjoint feasible solutions $\mathcal{F}$ and $\mathcal{F}'$ of any instance in $\mathcal{C}$ there exists an f-separable (planar) graph G with node set $\mathcal{F} \cup \mathcal{F}'$ with the following *exchange* property. For each element $u \in U$ that is covered both by $\mathcal{F}$ and $\mathcal{F}'$, there is an edge (S, S') in G with $S \in \mathcal{F}$ and $S' \in \mathcal{F}'$ with $u \in S \cap S'$.

▶ **Theorem 2.** *For any non-decreasing strictly sublinear function f, every f-separable class of MC instances (closed under removing elements and sets) admits a PTAS via Algorithm 1.*

Applications. We now describe several problems which are special instances of the MC problem. Then, in Theorem 3, we state several PTASs for each of these problems that can be obtained from our analysis of local search. As this latter part is essentially a direct consequence of Theorem 2, the details will be provided in the full version.

▶ **Problem 1.** *Let H be a set of ground elements, $\mathcal{S} \subseteq 2^H$ be a set of ranges and k be a positive integer. A range $S \in \mathcal{S}$ is hit by a subset H' of H if $S \cap H' \neq \emptyset$. The* Maximum Hitting *(MH) problem asks for a k-subset H' of H such that the number of ranges hit by H' is maximized.*

▶ **Problem 2.** *Let $G = (V, E)$ be a graph and k be a positive integer. A vertex $v \in V$ dominates itself and all its neighbors. The* Maximum Dominating *(MD) problem asks for a k-subset V' of V such that the number of vertices dominated by V' is maximized.*

▶ **Problem 3.** *Let T be a* 1.5D *terrain – an x-monotone polygonal chain in $\mathbb{R}^2$ consisting of a set of vertices $\{v_1, v_2, \ldots, v_m\}$ sorted in increasing order of their x-coordinate, and v_i and v_{i+1} are connected by an edge for all $i \in [m-1]$. For any two points $x, y \in T$, we say that y guards x if each point in $\overline{xy}$ lies above or on the terrain. Given finite sets $X, Y \subseteq T$ and a positive integer k, the* Maximum Terrain Guarding *(MTG) problem asks for a k-subset Y' of Y such that the number of points of X guarded by Y' is maximized.*

Let r be an even, positive integer. A set of regions in $\mathbb{R}^2$ (each bounded by a closed Jordan curve), is called *r-admissible* if for any two such regions q_1, q_2, the curves bounding them cross $s \leq r$ times for some even s and $q_1 \setminus q_2$ and $q_2 \setminus q_1$ are connected regions. A set of regions are called *pseudo-disks* if it is 2-admissible (e.g., a set of disks or squares).

The below consequences of Theorem 2 hold since the corresponding SC problem is known to be *planarizable* or by constructing the exchange graph as a minor of the input graph.

▶ **Theorem 3.** *Local search gives a PTAS for:*

(V) *the MVC problem on f-separable and subgraph-closed graph classes,*
(T) *the MTG problem.*

and the following classes of MC problems:
(C_1) *the set of ground elements is a set of points in $\mathbb{R}^3$, and the family of subsets is induced by a set of halfspaces in $\mathbb{R}^3$.*
(C_2) *the set of ground elements is a set of points in $\mathbb{R}^2$, and the family of subsets is induced by a set of convex pseudodisks (a set of convex objects where any two objects can have at most two intersections in their boundary).*

and the following MH problems:
(H_1) *the set of ground elements is a set of points in $\mathbb{R}^2$, and the set of ranges is induced by a set of r-admissible regions (this includes pseudodisks, same-height axis-parallel rectangles, circular disks, translates of convex objects).*
(H_2) *the set of ground elements is a set of points in $\mathbb{R}^3$, and the set of ranges is induced by a set of halfspaces in $\mathbb{R}^3$.*

and MD problems in each of the following graph classes:
(D_1) *intersection graphs of homothetic copies of convex objects (which includes arbitrary squares, regular k-gons, translated and scaled copies of a convex object).*
(D_2) *non-trivial minor-closed graph classes.*

2 Color-Balanced Divisions

In this section we provide the main tool (see Theorem 7) used to prove our main result (i.e., Theorem 2). We first describe a new subtle strengthening (see Lemma 5) of the standard division theorem on f-separable graph classes (see Theorem 4). This builds on the concept of $(r, f(r))$-divisions (in the sense of Henzinger et al. [23]) of graphs in an f-separable graph class. We then extend this strengthened division lemma by suitably aggregating the pieces of the partition to obtain a *two-color balanced* version (see Theorem 7). This result generalizes to more than two colors. However, as our applications stem from the two-colored version, we defer the generalization to the full version.

Frederickson [17] introduced the notion of an r-*division* of an n-vertex graph G, namely, a cover of $V(G)$ by $\Theta(\frac{n}{r})$ sets each of size $O(r)$ where each set has $O(\sqrt{r})$ *boundary* vertices, i.e., $O(\sqrt{r})$ vertices in common with the other sets. Frederickson showed that, for any r, every planar graph G has an r-division and that one can be computed in $O(n \log n)$ time. This result follows from a recursive application of the Lipton-Tarjan planar separator theorem [27]. This notion was further generalized by Henzinger et al. [23] to $(r, f(r))$-divisions[‡] where f is a function in $o(r)$ and each set has at most $f(r)$ vertices in common with the other sets. They noted that Frederickson's proof can easily be adapted to obtain an $(r, c \cdot f(r))$-division of any graph G from a subgraph-closed f-separable graph class – as formalized in Theorem 4. Note that we use an equivalent but slightly different notation than Frederickson and Henzinger et al. in that we consider the "boundary" vertices as a single separate set apart from the non-boundary vertices in each "region", i.e., our *divisions* are actually partitions of the vertex set. This allows us to carefully describe the number of vertices inside each "region".

[‡] They use a more general notion of (r, s)-division but we need the restricted version as described here.

▶ **Theorem 4** ([17, 23])**.** *For any subgraph-closed f-separable class of graphs $\mathcal{G}$, there are constants c_1, c_2 such that every graph G in the class has an $(r, c_1 \cdot f(r))$-*division *for any r. Namely, for any $r \geq 1$, there is an integer $t \in \Theta(\frac{n}{r})$ such that V can be partitioned into $t+1$ sets $\mathcal{X}, V_1, \ldots, V_t$ where the following properties hold.*

(i) $N(V_i) \cap V_j = \emptyset$ *for each* $i \neq j$ *and* $\mathcal{X} = \bigcup_i N(V_i)$*,*
(ii) $|V_i \cup N(V_i)| \leq r$ *for each* i*,*
(iii) $|N(V_i)| \leq c_1 \cdot f(r)$ *for each* i *(thus,* $|\mathcal{X}| \leq \sum_{i=1}^{t} |N(V_i)| \leq c_2 \cdot \frac{f(r) \cdot n}{r}$*).*

Moreover, such a partition can be found in $O(g(n) \log n)$ time where $g(n)$ is the time required to find an f-separation in $\mathcal{G}$.

We obtain our color-balanced version of Theorem 4 via two steps. First, we strengthen the notion of $(r, f(r))$-divisions to *uniform* $(r, f(r))$-divisions. A *uniform* $(r, f(r))$*-division* is an $(r, f(r))$-division where the $\Theta(\frac{n}{r})$ sets have a *uniform* size. Namely, there are not only $O(r)$ many **internal** vertices (as in Theorem 4) but there are also $\Omega(r)$ many of them.

It is important to note that while this uniformity condition (i.e., that each region is not *too small* – see Lemma 5(ii)) has **not** been needed in the past[§], it is essential for our analysis of local search as applied to MC problems in the next section. Moreover, to the best of our knowledge, neither Frederickson's construction nor more modern constructions (e.g. [25]) of an r-division explicitly guarantee that the resulting r-division is uniform. To be specific, Frederickson's approach consists of two steps. The first step recursively applies the separator theorem until each region together with its boundary is "small enough". In the second step, each region where the boundary is "too large" is further divided. This is accomplished by applying the separator theorem to a weighted version of each such region where the boundary vertices are uniformly weighted and the non-boundary vertices are zero-weighted. Clearly, even a single application of this latter step may result in regions with $o(r)$ interior vertices. Modern approaches (e.g. [25]) similarly involve applying the separator theorem to weighted regions where boundary vertices are uniformly weighted and interior vertices are zero-weighted, i.e., regions which are *too small* are not explicitly avoided.

In the second step, we generalize the uniform $(r, f(r))$-divisions to *two-color uniform* $(r, f(r))$*-divisions* of a two-colored graph (the coloring need not be proper in the usual sense). A two-color uniform $(r, f(r))$-division of a two-colored graph is a uniform $(r, f(r))$-division where each set has the "same" proportion of each color class (this is formalized in Theorem 7).

The remainder of this section is outlined as follows. We will first show that for every f-separable graph class $\mathcal{G}$ there is a constant c such that every graph in $\mathcal{G}$ has a uniform $(r, c \cdot f(r))$-division (see Lemma 5). We then use this result to show that for every f-separable graph class $\mathcal{G}$ there is a constant c' such that every two-colored graph in $\mathcal{G}$ has a two-color uniform $(rq, c' \cdot q \cdot f(r))$-division for any q – see Theorem 7. Our proofs are constructive and lead to efficient algorithms that produce such divisions when there is a corresponding efficient algorithm to compute an f-separation.

To prove the first result, we start from a given $(r, f(r))$-division and "group" the sets carefully to obtain the desired uniformity. For the two-colored version, we start from a uniform $(r, f(r))$-division and again regroup the sets via a reformulation of the problem as a partitioning problem on two-dimensional vectors. Lemma 6 is used for this regrouping.

▶ **Lemma 5.** *Let $\mathcal{G}$ be a subgraph-closed f-separable graph class and G be a sufficiently large n-vertex graph in $\mathcal{G}$. There are constants r_0, x_0 (depending only on f) such that for any*

[§] E.g., to analyse local search for SC problems [30], or for fast algorithms to find shortest paths [23].

$r \in [r_0, \frac{n}{x_0}]$ there is an integer $t \in \Theta(\frac{n}{r})$ such that $V(G)$ can be partitioned into $t+1$ sets $\mathcal{X}, V_1, \dots, V_t$ where c_1, c_2 are constants (depending only on f) with the following properties.

(i) *$N(V_i) \cap V_j = \emptyset$ for each $i \neq j$ and $\mathcal{X} = \bigcup_i N(V_i)$,*

(ii) *$\boxed{|V_i| \geq \frac{r}{2}}$¶ and $|V_i| \leq 2r$ for each i,*

(iii) *$|N(V_i)| \leq c_1 \cdot f(r)$ for each i (thus, $|\mathcal{X}| \leq \sum_{i=1}^{t} |N(V_i)| \leq \frac{c_2 \cdot f(r) \cdot n}{r}$).*

Moreover, such a partition can be found in $O(h(n) + n)$ time where $h(n)$ is the amount of time required to produce an $(r, f(r))$-division of G.

Proof. We pick the constants c_1', c_2' as obtained from Theorem 4 for our f-separable subgraph-closed graph class. We further pick r_0 such that it is divisible by 8 and $c^* = 1 - c_2' \cdot f(\frac{r_0}{8}) \cdot (\frac{r_0}{8})^{-1} > 0$. Now, let $x_0 = \frac{3}{c^*}$, and assume $r \in [r_0, \frac{n}{x_0}]$ in what follows.

Consider an $(\lfloor\frac{r}{8}\rfloor, c_1' \cdot f(\lfloor\frac{r}{8}\rfloor))$-division $\mathcal{U} = (\mathcal{X}, U_1, \dots, U_\ell)$ as given by Theorem 4 – note: $|\mathcal{X}| \leq \frac{c_2' \cdot f(\lfloor\frac{r}{8}\rfloor) n}{\lfloor\frac{r}{8}\rfloor}$. We further define c_ℓ so that $\ell = c_\ell \cdot \frac{8 \cdot n}{r}$. We will partition $[\ell]$ into t sets $I_1, \dots, I_t$ such that $(\mathcal{X}, V_1, \dots, V_t)$ is a uniform $(r, c \cdot f(r))$-division $\mathcal{X}, V_1, \dots, V_t$ where $V_i = \bigcup_{j \in I_i} U_j$. In order to describe the partitioning, we first observe some useful properties of $U_1, \dots, U_\ell$ where, without loss of generality, $|U_1| \geq \dots \geq |U_\ell|$. Let $n^* = \sum_{j=1}^{\ell} |U_j|$, and set $t = \lceil\frac{n^*}{r}\rceil$. Note that:

$$n^* = \sum_{j=1}^{\ell} |U_j| = n - |\mathcal{X}| \geq n \cdot \left(1 - \frac{c_2' \cdot f(\lfloor\frac{r}{8}\rfloor)}{\lfloor\frac{r}{8}\rfloor}\right).$$

From our choice of t, the average size of the sets V_i is $\frac{n^*}{t} \in (\frac{r}{1+\frac{r}{n^*}}, r]$. Additionally, $n^* \geq c^* \cdot n$, i.e., $c^* \leq \frac{n^*}{n}$. Thus, we have $r \leq \frac{n^*}{3}$, and the average size of our sets $|V_i|$ is in $[\frac{3r}{4}, r]$.

Notice that $\frac{\ell}{t} \leq c_\ell \cdot \frac{8 \cdot n}{r} \cdot (\frac{n^*}{r})^{-1} \leq \frac{8c_\ell}{c^*}$. We build the sets I_i such that $|I_i| \leq 40 \cdot \frac{c_\ell}{c^*}$. This provides $|N(V_i)| \leq 40 \cdot \frac{c_\ell}{c^*} \cdot c_1' f(\lfloor\frac{r}{8}\rfloor) \in O(f(r))$. We build the sets I_i in two steps. In the first step we greedily fill the sets I_i according to the largest unassigned set U_j as follows. For each j^* from 1 to ℓ, we consider an index $i^* \in [t]$ where $|I_{i^*}| < 32 \cdot \frac{c_\ell}{c^*}$ and $|V_{i^*}|$ is minimized. If $|V_{i^*}| \leq \frac{n^*}{t}$, then we place j^* into I_{i^*}, that is, we replace V_{i^*} with $V_{i^*} \cup U_{j^*}$. Otherwise (there is no such index i^*), we proceed to step two (below). Before discussing step two, we first consider the state of the sets V_i at the moment when this greedy placement finishes. To this end, let j^* be the index of the first (i.e., the largest) U_j which has not been placed.

Claim 1: *If $|V_i| \leq \frac{n^*}{t}$ for every i, then each set U_j has been merged into some V_i and the V_i's satisfy the conditions of the lemma.*

First, suppose there is an unallocated set U_j. Since $|V_i| \leq \frac{n^*}{t}$ for each $i \in [t]$, our greedy procedure stopped due to having $|I_i| = 32 \cdot \frac{c_\ell}{c^*}$ for each $i \in [t]$. This contradicts the average size of the I_i's being $\frac{\ell}{t} \leq 8 \cdot \frac{c_\ell}{c^*}$. So, every set U_j must have been merged into some V_i. Thus, since $|V_i| \leq \frac{n^*}{t}$ and the average of the $|V_i|$'s is $\frac{n^*}{t}$, we have that for every $i \in [t]$, $|V_i| = \frac{n^*}{t}$. Moreover, for each $i \in [t]$, $|I_i| \leq 32\frac{c_\ell}{c^*}$. Thus the V_i's satisfy the lemma.

Claim 2: *For every $i \in [t]$, $|V_i| \geq \frac{r}{2}$.*

Suppose some index i has $|V_i| < \frac{r}{2}$. Notice that, if $|I_i| < 32 \cdot \frac{c_\ell}{c^*}$, then for every $i' \in [t]$, $|V_{i'}| \leq |V_i| + \frac{r}{8} \leq \frac{3r}{4} \leq \frac{n^*}{t}$, i.e., contradicting Claim 1. Thus, $|I_i| = 32 \cdot \frac{c_\ell}{c^*}$ for each $i \in [t]$ where $|V_i| < \frac{r}{2}$. For each $i' \in [t], j' \in [\ell]$, let $I_{i'}^{j'}$ and $V_{i'}^{j'}$ be the states of $I_{i'}$ and $V_{i'}$(respectively) directly after index j' has been added to some set $I_{i''}$ by the greedy algorithm.

¶ This lower bound is the difference from the known Theorem 4.

We now let $\hat{j}$ be the largest index in I_i, and assume (without loss of generality) that for every $i' \in [t] \setminus \{i\}$, if $|V_{i'}^{\hat{j}}| < \frac{r}{2}$, then $I_{i'}^{\hat{j}} < 32 \cdot \frac{c_\ell}{c^*}$. Intuitively, i is the "first" index which attains $|I_i| = 32 \cdot \frac{c_\ell}{c^*}$ while still having $|V_i| < \frac{r}{2}$. Now, since $|I_i^{\hat{j}}| = 32 \cdot \frac{c_\ell}{c^*}$, and $|V_i^{\hat{j}}| < \frac{r}{2}$, we have $|U_{\hat{j}}| < r \cdot \frac{c^*}{64c_\ell}$. Thus, for every iteration $j > \hat{j}$, we have $|U_j| < r \cdot \frac{c^*}{64c_\ell}$. This means that after iteration $\hat{j}$, the number of unallocated vertices is strictly less than:

$$\sum_{j=\hat{j}}^{\ell} U_j < \ell \cdot r \cdot \frac{c^*}{64c_\ell} \leq t \cdot 8 \cdot \frac{c_\ell}{c^*} \cdot r \cdot \frac{c^*}{64c_\ell} = \frac{tr}{8}.$$

In particular, this means that on average each set V_i can grow by less than $\frac{r}{8}$. However, due to our choice of i, we see that for every $i' \in [t] \setminus \{i\}$, $|V_{i'}^{\hat{j}}| \leq |V_i^{\hat{j}}| + \frac{r}{8} < \frac{r}{2} + \frac{r}{8}$. This means that even if we allocate all the remaining vertices, the average size of our sets V_i will be strictly less than $\frac{3r}{4} \leq \frac{n^*}{t}$, i.e., providing a contradiction and proving Claim 2.

Claim 3: *If every $j \in [\ell]$ is placed into some I_i, the V_i's satisfy the conditions of the lemma.* First, note that $|I_i|$ is at most $32 \cdot \frac{c_\ell}{c^*}$, i.e., $|N(V_i)| \in O(f(r))$. By Claim 2, we see that $|V_i| \geq \frac{r}{2}$ for each $i \in [t]$. Additionally, from the greedy construction, we have that $|V_i| \leq \frac{n^*}{t} + \frac{r}{8}$. Thus, $|V_i| \in [\frac{r}{2}, \frac{9r}{8}] \subset [\frac{r}{2}, 2r]$.

We now describe the second step. By Claim 3, we assume there are unassigned sets U_j. By Claim 2, for every $i \in [t]$, $|V_i| \geq \frac{r}{2}$. Finally, by Claim 1, there is an index i' where $|V_{i'}| > \frac{n^*}{t}$. Thus, since we have $t = \lceil \frac{n^*}{r} \rceil$ sets which partition at most n^* elements, there must be some index i'' where $|V_{i''}| \leq \frac{n^*}{t}$ and $|I_{i''}| = 32 \cdot \frac{c_\ell}{c^*}$, i.e., $|U_{j^*}| \leq \frac{n^*}{t} \cdot (32 \cdot \frac{c_\ell}{c^*})^{-1} \leq \frac{r \cdot c^*}{32 \cdot c_\ell}$ where U_{j^*} is the largest unassigned set. Notice that there are at most $\ell \leq t \cdot 8 \cdot \frac{c_\ell}{c^*}$ indices which can be assigned and all the remaining sets contain at most $|U_{j^*}|$ vertices. If we spread these remaining U_j's uniformly throughout our V_i's, we will place at most $8 \cdot \frac{c_\ell}{c^*} \cdot |U_{j^*}| \leq \frac{r}{4}$ vertices into each V_i. Thus, for each $i \in [t]$, we have $|V_i| \leq \frac{n^*}{t} + \frac{r}{8} + \frac{r}{4} \leq 2r$. So, by uniformly assigning these remaining indices, we have $|I_i| \leq 40 \cdot \frac{c_\ell}{c^*}$, $|V_i| \in [\frac{r}{2}, 2r]$, and $|N(V_i)| \leq 40 \cdot \frac{c_\ell}{c^*} \cdot c_1' f(\lfloor \frac{r}{8} \rfloor) \in O(f(r))$, as needed.

We conclude with a brief discussion of the time complexity. First, we generate the $(\lfloor \frac{r}{8} \rfloor, c_1' f(\lfloor \frac{r}{8} \rfloor))$-division in $h(n)$ time. We then sort the sets $|U_1| \geq \ldots \geq |U_\ell|$ (this can be done in $O(n)$ time via bucket sort). In the next step we greedily fill the index sets – this takes $O(n)$ time. Finally, we place the remaining "small" sets uniformly throughout the V_i's – taking again $O(n)$ time. Thus, we have $O(h(n) + n)$ time in total. ◀

We now prove a technical lemma which, together with the previous lemma regarding uniform divisions, provides our uniform two-color balanced divisions (see Theorem 7).

▶ **Lemma 6.** *Let c and c' be positive constants, and $A = \{(a_1, b_1), \ldots, (a_n, b_n)\} \subseteq (\mathbb{Q} \cap [0, \infty))^2$ be a set of 2-dimensional vectors where $a_i + b_i \in [c', c]$ for each $i \in [n]$, and $\alpha \in [0, 1]$ such that $\sum_{i=1}^{n} a_i = \alpha \cdot \sum_{i=1}^{n} b_i$. Then:*

($\star$) *There is a permutation $p_1, \ldots, p_n$ of $[n]$ such that for any $1 \leq i \leq i' \leq n$, $|\sum_{j=i}^{i'} (a_{p_j} - \alpha \cdot b_{p_j})| \leq 2 \cdot c$; and*

($\star\star$) *For any positive integer q' such that $q'(q'+1) \leq n$ there is a partitioning of $[n]$ into subsets $I_1, \ldots, I_k$ such that for each $j \in [k]$:*

- **(i)** *$q' \leq |I_j| \leq q' + 1$ (thus, $q'c' \leq \sum_{i \in I_j} a_i + b_i \leq q'c + c$),*
- **(ii)** *$|\sum_{i \in I_j} (a_i - \alpha \cdot b_i)| \leq 2 \cdot c$.*

Moreover, the permutation $p_1, \ldots, p_n$ and partition can be computed in $O(n)$ time.

Proof. We first prove ($\star$). To this end, we partition $[n]$ into three sets $A_{>0}$, $A_{<0}$, and $A_{=0}$ according to whether the *weighted* difference $d_i = a_i - \alpha \cdot b_i$ is positive, negative, or 0 (respectively). Note that, $\sum_{i=1}^{n} d_i = 0$ and for each $i \in [n]$, $|d_i| \leq c$. We pick indices one-by-one from $A_{>0}$, $A_{<0}$, $A_{=0}$ to form the permutation.

We now construct a permutation $p_1, \ldots, p_n$ on the indices $[n]$ so that any consecutive subsequence S has $|\sum_{i \in S} d_{p_i}| \leq 2 \cdot c$. For notational convenience, for each $j \in [n]$, we use $\delta_{<j}$ to denote $\sum_{i=1}^{j-1} d_{p_i}$. We now pick the p_i's so that for each j, $|\delta_{<j}| \leq c$. We initialize $\delta_{<1} = 0$. For each j from 1 to n we proceed as follows. Assume that $|\delta_{<j}| \leq c$. We further assume that any index $i \in \{p_1, \ldots, p_{j-1}\}$ has been removed from the sets $A_{>0}$, $A_{<0}$, and $A_{=0}$. If $\delta_{<j}$ is negative, $A_{>0}$ must contain an index j^* since $\sum_{i \in [n]} d_i = 0$. Moreover, if we set $p_j = j^*$, we have $|\delta_{<j+1}| \leq c$ as needed (we also remove the index j^* from $A_{>0}$ at this point). Similarly, if $\delta_{<j}$ is positive, we pick any index j^* from $A_{<0}$, remove it from $A_{<0}$, and set $p_j = j^*$. Finally, when $\delta_{<j} = 0$, we take any index j^* from $A_{>0} \cup A_{<0} \cup A_{=0}$, remove it from $A_{>0} \cup A_{<0} \cup A_{=0}$, and set $p_j = j^*$. Thus, in all cases, $|\delta_{<j+1}| \leq c$. Notice that, for any $1 \leq j \leq j' \leq n$, we have $|\sum_{i=j}^{j'} d_{p_i}| = |\delta_{<j} - \delta_{<j'+1}| \leq |\delta_{<j}| + |\delta_{<j'+1}| \leq 2 \cdot c$ (thus, establishing ($\star$)).

We now prove ($\star\star$) using ($\star$). We partition $[n]$ to form the sets $I_1, \ldots, I_k$ by splitting $p_1, \ldots, p_n$ into k consecutive subsequences of almost equal size. Namely, since $\frac{n}{q'} - \frac{n}{q'+1} \geq 1$, we can pick a positive integer $k \in \left[\frac{n}{q'+1}, \frac{n}{q'}\right]$. Then $q'k \leq n \leq (q'+1)k$, so we can make the sets $I_1, \ldots, I_{n-q'k}$ with $q'+1$ indices each and the sets $I_{n-q'k+1}, \ldots, I_k$ with q' indices each by partitioning π into these sets in order. This is all easily accomplished in $O(n)$ time. ◀

We will now use Lemmas 5 and 6 to prove Theorem 7. In particular, for a given two-colored graph G where G belongs to an f-separable graph class, we first construct a uniform $(r, c \cdot f(r))$-division $(\mathcal{X}, V_1, \ldots, V_t)$ of G as in Lemma 5. From this division we can again carefully combine the V_i's to make new sets W_j where each W_j has roughly the same size and contains roughly the same proportion of each color class as occurring in G. This follows by simply imagining each region V_i of the uniform $(r, c \cdot f(r))$-division as a two-dimensional vector (according to its coloring) and then applying Lemma 6.

▶ **Theorem 7.** *Let $\mathcal{G}$ be a subgraph-closed f-separable graph class and G be a 2-colored n-vertex graph in $\mathcal{G}$ with color classes Γ_1, Γ_2 such that $|\Gamma_2| \geq |\Gamma_1|$. For any q and $r \ll n$ where r is suitably large, there is an integer $t \in \Theta(\frac{n}{q \cdot r})$ such that V can be partitioned into $t+1$ sets $\mathcal{X}, V_1, \ldots, V_t$ where c_1, c_2 are constants (depending only on f) and there is an integer $q' \in [q, 2q-1]$ satisfying the following properties.*

(i) *$N(V_i) \cap V_j = \emptyset$ for each $i \neq j$ and $\mathcal{X} = \bigcup_i N(V_i)$,*
(ii) *$|V_i| \geq \frac{q' \cdot r}{2}$ and $|V_i| \leq 2 \cdot (q'+1) \cdot r$ for each i,*
(iii) *$|N(V_i)| \leq c_1 \cdot q \cdot f(r)$ for each i*
(thus, $|\mathcal{X}| \leq \sum_{i=1}^{t} |N(V_i)| \leq \frac{c_2 \cdot f(r) \cdot n}{r}$),
(iv) *$\left| |V_i \cap \Gamma_1| - \frac{|\Gamma_1|}{|\Gamma_2|} \cdot |V_i \cap \Gamma_2| \right| \leq 4 \cdot r$ for each i.*

Moreover, such a partition can be found in $O(h(n) + n)$ time where $h(n)$ is the amount of time required to produce a uniform $(r, c \cdot f(r))$-division of G.

3 Proof of Theorem 2: PTAS for f-Separable Maximum Coverage

Recall that, we have an f-separable instance of MC where f is strictly sublinear. Our algorithm is based on local search. We fix a sufficiently large positive constant integer $b \geq 1$. Given an f-separable instance of MC, we pick an arbitrary initial solution $\mathcal{A}$. We check if it is possible to replace $2b^2+2b$ sets in $\mathcal{A}$ with the same number of sets from $\mathcal{F}$ so that the total number of elements covered is increased. We perform such a replacement (swap) as long as there is one. We stop if there is no profitable swap and output the resulting solution.

We will show that for sufficiently large b the above algorithm yields a $(1-8c_1f(b)/b)$-approximate solution and that it runs in polynomial time (for constant b). Here, c_1 is the constant from Theorem 7. Setting b to be sufficiently large will complete this proof. Note that, if $c_1<1$, Theorem 7 also holds for $c_1=1$. Thus, it suffices to consider $c_1\geq 1$.

Since each step increases the number of covered elements, the number of iterations of the above algorithm is at most $|U|$. In each iteration, we consider each of the $\binom{k}{b}\binom{|\mathcal{F}|}{b}$ potential b-swaps, and check whether it is an improvement. Therefore, the total running time of the algorithm is polynomial for constant b.

It remains to establish the performance guarantee. Let $\mathcal{O}$ be an optimum solution to the instance and let $\mathcal{A}$ be the (locally optimal) solution output by the algorithm. Let OPT, ALG denote the number of elements covered by $\mathcal{O}$, $\mathcal{A}$, respectively.

Suppose that ALG $<\left(1-\frac{8c_1f(b)}{b}\right)$ OPT. We will show that implies that there is a profitable swap (contradicting the local optimality of $\mathcal{A}$ and hence completing the proof).

We claim that it suffices to consider the case when $\mathcal{O},\mathcal{A}$ are disjoint, which is justified as follows. Assume that $\mathcal{O}\cap\mathcal{A}\neq\emptyset$. We remove the sets in $\mathcal{O}\cap\mathcal{A}$ from $\mathcal{F}$ and all the *elements* covered by these sets from U. Moreover, we decrease k by $|\mathcal{O}\cap\mathcal{A}|$ and replace $\mathcal{O}$ with $\mathcal{O}\setminus\mathcal{A}$ and $\mathcal{A}$ with $\mathcal{A}\setminus\mathcal{O}$. Since our class of instances is closed under removing sets and elements, the resulting instance is still contained in the class. Moreover, $|\bigcup\mathcal{A}|<\left(1-\frac{8c_1f(b)}{b}\right)|\bigcup\mathcal{O}|$ (note that the number of elements covered by $\mathcal{A}$ and $\mathcal{O}$, respectively, decreases by the same amount as we remove the elements covered by $\mathcal{A}\cap\mathcal{O}$ from the instance). Finally, a feasible and profitable swap in the reduced instance implies that the same swap is also feasible and profitable in the original.

Therefore, we assume from now on that $\mathcal{A}$ and $\mathcal{O}$ are disjoint. Since our instance is f-separable, there exists an f-separable graph G with precisely $2k$ nodes for the two feasible solutions $\mathcal{O}$ and $\mathcal{A}$ with the properties stated in Definition 1.

We now apply Theorem 7 to G with color classes $\Gamma_1=\mathcal{O}$ and $\Gamma_2=\mathcal{A}$ and with parameters $r=b$ and $q=b$. Here, we assume that the constant b has been picked sufficiently large as required by Theorem 7. We further assume that the number $2k$ of nodes in the exchange graph is substantially larger than b as the problem can be solved to optimality in polynomial time for constant k. Since $|\mathcal{O}|=|\mathcal{A}|=k$, the two color classes in G are perfectly balanced. Let $\mathcal{A}_i=\mathcal{A}\cap V_i$, $\mathcal{O}_i=\mathcal{O}\cap V_i$, $N_i^{\mathcal{O}}=N(V_i)\cap\mathcal{O}$ and $\bar{\mathcal{O}}_i=\mathcal{O}_i\cup N_i^{\mathcal{O}}$ for any part V_i with $i\in[t]$ of the resulting subdivision of G. Note that every set in $\mathcal{O}$ is in $\bar{\mathcal{O}}_i$ for some $i\in[t]$.

The idea of the analysis is to consider for each $i\in[t]$ a nearly balanced (but possibly infeasible) *candidate swap* that replaces in $\mathcal{A}$ the sets $\mathcal{A}_i$ with $\bar{\mathcal{O}}_i$. We will first show that there exists a profitable candidate swap if ALG $<\left(1-\frac{8c_1f(b)}{b}\right)$ OPT. Second, we will show that even a *feasible* profitable swap can be obtained by adding only some of the sets in $\bar{\mathcal{O}}_i$ as the candidate swap was nearly balanced.

For technical reasons we will define a set Z of elements that we (temporarily) disregard from our calculations because they will remain covered and thus do not impact which of the sets in $\bar{\mathcal{O}}_i$ we will pick for the feasible swap. Let $Z_i=\bigcup(\mathcal{A}\setminus\mathcal{A}_i)$ be the set of elements that remain covered even if $\mathcal{A}_i$ is removed from $\mathcal{A}$ and let $Z=\bigcap_{i=1}^t Z_i$ be the set of elements that are covered by $\mathcal{A}$ but not exclusively by one of the $\mathcal{A}_i$. In particular, Z contains all elements in $\bigcup(\mathcal{X}\cap\mathcal{A})$. Let $L_i=\bigcup\mathcal{A}_i\setminus Z$ be the set of elements that are "lost" when removing the $\mathcal{A}_i$ from $\mathcal{A}$. Moreover, let $W_i=\bigcup\bar{\mathcal{O}}_i\setminus Z_i$ be the set of elements that are "won" when we add all the sets of $\bar{\mathcal{O}}_i$ after removing $\mathcal{A}_i$.

We claim that $\sum_{i=1}^t |L_i| \le \mathrm{ALG} - |Z|$. To this end, note that $Z \subseteq \bigcup \mathcal{A}$ and that the family $\{L_i\}_{i\in[t]}$ contains pairwise disjoint sets because all elements that are not exclusively covered by a single $\mathcal{A}_i$ are contained in Z and thus removed.

We further claim that $\sum_{i=1}^t |W_i| \ge \mathrm{OPT} - |Z|$. To see this, note first that every element in Z contributes 0 to the left hand side and 0 or -1 to the right hand side. Every element covered by $\mathcal{O}$ but not by $\mathcal{A}$ contributes at least 1 to the left hand side and precisely 1 to the right hand side. Finally, consider an element u that is covered both by $\mathcal{A}$ and by $\mathcal{O}$ but does not lie in Z. Since $u \notin Z$ there is no $S \in \mathcal{X} \cap \mathcal{A}$ covering u. This also implies that u lies in a set $S \in \mathcal{A}_i$ for some *unique* $i \in [t]$. Thus, we even have $u \notin Z_i$. Since G is an exchange graph, there is some set $T \in \mathcal{O}$ with $u \in T$ and some set $S' \in \mathcal{A}$ with $u \in S'$ such that S' and T are adjacent in G. Since $u \notin Z_i$ we have $S' \in \mathcal{A}_i$. Since T is adjacent to $S' \in \mathcal{A}_i \subseteq V_i$, we have $T \in N(V_i)$. Since $T \in \mathcal{O}$, it follows that $T \in \mathcal{N}_i^{\mathcal{O}} \subseteq \bar{\mathcal{O}}_i$. Thus $u \in \bigcup \bar{\mathcal{O}}_i \setminus Z_i = W_i$. Hence u contributes at least 1 to the left hand side and precisely 1 to the right hand side of $\sum_{i=1}^t |W_i| \ge \mathrm{OPT} - |Z|$, which shows the claim.

By $\mathrm{OPT} > \mathrm{ALG} \ge |Z|$, $\min\limits_{\substack{i\in[t]\\ |W_i|>0}} \dfrac{|L_i|}{|W_i|} \le \dfrac{\sum_{i=1}^t |L_i|}{\sum_{i=1}^t |W_i|} \le \dfrac{\mathrm{ALG} - |Z|}{\mathrm{OPT} - |Z|} \le \dfrac{\mathrm{ALG}}{\mathrm{OPT}} < 1 - \dfrac{8c_1 f(b)}{b}$.

Hence, we pick $i \in [t]$ so that $\dfrac{|L_i|}{|W_i|} < 1 - \dfrac{8c_1 f(b)}{b}$. (1)

Recall that $c_1 \ge 1$ and assume that b is large enough so that $f(b) \ge 1$. Then by Properties (ii), (iii), and (iv) of Theorem 7 (respectively), we have that $|V_i| \ge b^2/2$, $|N(V_i)| \le c_1 b \cdot f(b)$, and $||\mathcal{A}_i| - |\mathcal{O}_i|| \le 4b$ (respectively). Now, since $|\mathcal{A}_i| + |\mathcal{O}_i| = |V_i|$, we have $|\bar{\mathcal{O}}_i| \le \frac{1}{2}|V_i| + 2b + c_1 b \cdot f(b)$ and $|\mathcal{A}_i| \ge \frac{1}{2}|V_i| - 2b$. Hence

$$\frac{|\mathcal{A}_i|}{|\bar{\mathcal{O}}_i|} \ge \frac{\frac{1}{2}|V_i| - 2b}{\frac{1}{2}|V_i| + 2b + c_1 b \cdot f(b)} \ge \frac{(\frac{1}{2}|V_i| + 2b + 4c_1c_2 b \cdot f(b)) - 4b - 4c_1c_2 b \cdot f(b)}{\frac{1}{2}|V_i| + 2b + 4c_1c_2 b \cdot f(b)} \tag{2}$$

$$\overset{|V_i|\ge b^2/2}{\ge} 1 - \frac{8c_1 f(b)}{b}. \tag{3}$$

We are now ready to construct our feasible and profitable swap. We inductively define an order $S_1, \dots, S_{|\bar{\mathcal{O}}_i|}$ on the sets in $\bar{\mathcal{O}}_i$ where we require that

$$S_j = \arg\max_{S\in\bar{\mathcal{O}}_i} \left| S \setminus \left(Z_i \cup \bigcup_{\ell=1}^{j-1} S_\ell \right) \right| \quad \text{for any } j = 1, \dots, |\bar{\mathcal{O}}_i| \text{ where } S_1 \text{ maximizes } |S \setminus Z_i|.$$

Consider the following process of iteratively building a set W' starting with $W' = \emptyset$. Suppose that we add to W' the sets $(S_1 \setminus Z_i), \dots, (S_{|\bar{\mathcal{O}}_i|} \setminus Z_i)$ in this order ending up with $W' = W_i$. In doing so, the incremental gain is monotonically non-increasing due to the definition of the order on $\bar{\mathcal{O}}_i$ and due to the submodularity of the objective function. Hence,

for any prefix of the first j sets we have that $\left| \left(\bigcup_{\ell=1}^{j} S_\ell \right) \setminus Z_i \right| \ge \dfrac{j \cdot |W_i|}{|\bar{\mathcal{O}}_i|}$. (4)

Suppose that $|\bar{\mathcal{O}}_i| > |\mathcal{A}_i|$ (otherwise due to (1), we can just add all sets in $\bar{\mathcal{O}}_i$). Consider the swap where we replace the $|\mathcal{A}_i| \le 1/2|V_i| + 2b \le 2b^2 + 2b$ many sets $\mathcal{A}_i$ from the local optimum $\mathcal{A}$ with at most $|\mathcal{A}_i|$ many sets $\{S_1, \dots, S_{|\mathcal{A}_i|}\}$ from $\bar{\mathcal{O}}_i$.

We now analyze how this swap affects the objective function value. Notice that, by removing the sets in $\mathcal{A}_i$, the objective function value drops by

$$|L_i| \overset{(1)}{<} \left(1 - \frac{8c_1 f(b)}{b}\right) \cdot |W_i| \overset{(4)}{\leq} \left(1 - \frac{8c_1 f(b)}{b}\right) \frac{|\bar{O}_i|}{|\mathcal{A}_i|} \left| \left(\bigcup_{\ell=1}^{|\mathcal{A}_i|} S_\ell \right) \setminus Z_i \right|$$
$$\overset{(3)}{\leq} \left| \left(\bigcup_{\ell=1}^{|\mathcal{A}_i|} S_\ell \right) \setminus Z_i \right| .$$

The right hand side of this inequality is the increase of the objective function due to adding the sets $\{S_1, \ldots, S_{|\mathcal{A}_i|}\}$ after removing the sets in $\mathcal{A}_i$.

Therefore the above described swap is feasible and also profitable and thus $\mathcal{A}$ is not a local optimum leading to a contradiction (and completing the proof of Theorem 2).

References

1 Alexander A. Ageev and Maxim Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *J. Comb. Optim.*, 8(3):307–328, 2004. doi:10.1023/B:JOCO.0000038913.96607.c2.

2 Noga Alon, Paul Seymour, and Robin Thomas. A separator theorem for nonplanar graphs. *J. of the American Mathematical Society*, 3:801–808, 1990. doi:10.1090/S0894-0347-1990-1065053-0.

3 Rom Aschner, Matthew J. Katz, Gila Morgenstern, and Yelena Yuditsky. Approximation schemes for covering and packing. In *7th Int. Workshop on Algorithms and Computation (WALCOM'13)*, pages 89–100, 2013.

4 Ashwinkumar Badanidiyuru, Robert Kleinberg, and Hooyeon Lee. Approximating low-dimensional coverage problems. In *Symp. Computational Geometry (SoCG'12)*, pages 161–170, 2012. doi:10.1145/2261250.2261274.

5 Sayan Bandyapadhyay and Kasturi R. Varadarajan. On variants of k-means clustering. In *32nd Symp. Computational Geometry (SoCG'16)*, pages 14:1–14:15, 2016. doi:10.4230/LIPIcs.SoCG.2016.14.

6 Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995. doi:10.1007/BF02570718.

7 Sergio Cabello and David Gajser. Simple PTAS's for families of graphs excluding a minor. *Discrete Applied Mathematics*, 189:41–48, 2015.

8 Timothy M. Chan, Elyot Grant, Jochen Könemann, and Malcolm Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *23rd ACM-SIAM Symp. Discrete Algorithms (SODA'12)*, pages 1576–1585, 2012. URL: http://dl.acm.org/citation.cfm?id=2095116.2095241.

9 Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. In *25th ACM Symp. Computational Geometry (SoCG'09)*, pages 333–340, 2009. doi:10.1145/1542362.1542420.

10 Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012. doi:10.1007/s00454-012-9417-5.

11 Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local search yields approximation schemes for k-means and k-median in Euclidean and minor-free metrics. In *57th IEEE Symp. Foundations of Computer Science (FOCS'16)*, pages 353–364, 2016. doi:10.1109/FOCS.2016.46.

12 Vincent Cohen-Addad and Claire Mathieu. Effectiveness of local search for geometric optimization. In *31st Symp. Computational Geometry (SoCG'15)*, pages 329–343, 2015. doi:10.4230/LIPIcs.SOCG.2015.329.

13 Gérard Cornuéjols, George L. Nemhauser, and Laurence A. Wolsey. Worst-case and probabilistic analysis of algorithms for a location problem. *Operations Research*, 28(4):847–858, 1980. doi:10.1287/opre.28.4.847.

14 Minati De and Abhiruk Lahiri. Geometric dominating set and set cover via local search. *CoRR*, abs/1605.02499, 2016. URL: http://arxiv.org/abs/1605.02499, arXiv:1605.02499.

15 Thomas Erlebach and Erik Jan van Leeuwen. Approximating geometric coverage problems. In *19th ACM-SIAM Symp. Discrete Algorithms (SODA'08)*, pages 1267–1276, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347220.

16 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

17 Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987.

18 Zachary Friggstad, Mohsen Rezapour, and Mohammad R. Salavatipour. Local search yields a PTAS for k-means in doubling metrics. In *57th IEEE Symp. Foundations of Computer Science (FOCS'16)*, pages 365–374, 2016. doi:10.1109/FOCS.2016.47.

19 Matt Gibson and Imran A. Pirwani. Algorithms for dominating set in disk graphs: Breaking the log$n$ barrier - (extended abstract). In *18th European Symp. Algorithms (ESA'10)*, pages 243–254, 2010. doi:10.1007/978-3-642-15775-2_21.

20 Sathish Govindarajan, Rajiv Raman, Saurabh Ray, and Aniket Basu Roy. Packing and covering with non-piercing regions. In *24th European Symp. Algorithms (ESA'16)*, pages 47:1–47:17, 2016. doi:10.4230/LIPIcs.ESA.2016.47.

21 Sariel Har-Peled and Kent Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs. In *23rd European Symp. Algorithms (ESA'15)*, pages 717–728, 2015.

22 Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. Implementing data cubes efficiently. In *ACM SIGMOD Int. Conference on Management of Data (ICDM'96)*, pages 205–216, 1996. doi:10.1145/233269.233333.

23 Monika Rauch Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997. doi:10.1006/jcss.1997.1493.

24 R. B. O. Kerkkamp and Karen Aardal. A constructive proof of swap local search worst-case instances for the maximum coverage problem. *Oper. Res. Lett.*, 44(3):329–335, 2016. doi:10.1016/j.orl.2016.03.001.

25 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *45th Symp. Theory of Computing (STOC'13)*, pages 505–514, 2013. doi:10.1145/2488608.2488672.

26 Erik Krohn, Matt Gibson, Gaurav Kanade, and Kasturi R. Varadarajan. Guarding terrains via local search. *J. of Computational Geometry*, 5(1):168–178, 2014. URL: http://jocg.org/index.php/jocg/article/view/128.

27 Richard J Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM J. on Applied Mathematics*, 36(2):177–189, 1979. doi:10.1137/0136016.

28 Steffen Mecke and Dorothea Wagner. Solving geometric covering problems by data reduction. In *12th European Symp. Algorithms (ESA'04)*, pages 760–771, 2004. doi:10.1007/978-3-540-30140-0_67.

29 Nabil H. Mustafa, Rajiv Raman, and Saurabh Ray. Quasi-polynomial time approximation scheme for weighted geometric set cover on pseudodisks and halfspaces. *SIAM J. Comput.*, 44(1):1650–1669, 2015. doi:10.1137/14099317X.

30 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.

31 Erez Petrank. The hardness of approximation: Gap location. *Comput. Complexity*, 4:133–157, 1994. doi:10.1007/BF01202286.

32 Neil Robertson and Paul D Seymour. Graph minors. xx. wagner's conjecture. *J. of Combinatorial Theory, Series B*, 92(2):325–357, 2004. doi:10.1016/j.jctb.2004.08.001.

33 Kasturi R. Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In *42nd ACM Symp. Theory of Computing (STOC'10)*, pages 641–648, 2010. doi:10.1145/1806689.1806777.

Amortized Analysis of Asynchronous Price Dynamics

Yun Kuen Cheung[1]
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
ycheung@mpi-inf.mpg.de
https://orcid.org/0000-0002-9280-0149

Richard Cole[2]
Courant Institute, NYU, New York, USA
cole@cs.nyu.edu
https://orcid.org/0000-0002-5885-0222

Abstract

We extend a recently developed framework for analyzing asynchronous coordinate descent algorithms to show that an asynchronous version of tatonnement, a fundamental price dynamic widely studied in general equilibrium theory, converges toward a market equilibrium for Fisher markets with CES utilities or Leontief utilities, for which tatonnement is equivalent to coordinate descent.

2012 ACM Subject Classification Theory of computation → Market equilibria

Keywords and phrases Asynchronous Tatonnement, Fisher Market, Market Equilibrium, Amortized Analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.18

Related Version A full version is available at `https://arxiv.org/abs/1806.10952`.

Acknowledgements We thank several anonymous reviewers for their helpful suggestions.

1 Introduction

As is well known, it is PPAD-hard to compute equilibria for general games and markets [21, 9, 18, 8, 37, 10]. By viewing the players and the environment collectively as implicitly performing a computation, these hardness results indicate that, in general, a game or market cannot reach an equilibrium quickly (assuming no unexpected complexity results such as PPAD = FP). As a result, a lot of attention has been given to the design of polynomial-time algorithms to compute equilibria, either exactly or approximately, for specific families of games and markets. Most of these algorithms can be categorized as either simplex-like (e.g., Lemke-Howson [30]), numerical methods (e.g., the interior-point method [40] or the ellipsoid method [27]), or some carefully-crafted combinatorial algorithms (e.g., flow-based algorithms for computing an equilibrium of a market with agents having linear utility functions [22, 33, 24]).

[1] Part of the work done while this author was at the Courant Institute, NYU and at the Faculty of Computer Science, University of Vienna. He was supported in part by NSF Grant CCF-1217989, and the Vienna Science and Technology Fund (WWTF) project ICT10-002. Additionally the research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Grant Agreement no. 340506.

[2] This work was supported in part by NSF grant CCF-1527568.

© Yun Kuen Cheung and Richard Cole;
licensed under Creative Commons License CC-BY

However, it seems implausible that these algorithms describe the implicit computations in games or markets. In particular, many markets appear to have a highly distributed environment. This would appear to preclude computations which require centralized coordination, which is essential for the three categories of algorithms above. Consequently, in order to justify equilibrium concepts, we want natural algorithms which could plausibly be running (in an implicit form) in the associated distributed environments. Moreover, since it is preferable not to assume centralized timing or coordination, a desirable feature of such natural algorithms is robustness against asynchrony, which means such algorithms should remain effective even in situations where information transfer takes time and agents make decisions (i.e., perform computations) with possibly outdated information.

A first candidate for a natural algorithm in markets is tatonnement: it adjusts the price of a good upward if there is too much demand, and downward if too little. Indeed, tatonnement was proposed alongside the concept of a market equilibrium by Walras [38] in 1874. Since then, studies of market equilibria and tatonnement have received much attention in economics, operations research, and most recently in computer science; we list a small sample of the voluminous literature, focusing mainly on computer science works [2, 36, 23, 17, 19, 20, 14, 13, 35]. Underlying many of these works is the issue of what are plausible price adjustment mechanisms and in what types of markets they attain a market equilibrium.

The tatonnements studied in prior work have mostly been continuous, or discrete and synchronous. Cole and Fleischer [19] observed that real-world market dynamics are highly distributed and hence presumably asynchronous. They argued that any realistic price dynamics must involve out-of-equilibrium trade in order to induce the imbalances leading to price updates. Further, they argued that simple rules with relatively low information requirements were more plausible. The lowest imaginable level of information would be for each seller to only know the demand for the good it was selling, and for any price updating to occur in a non-coordinated manner, i.e., asynchronously. Accordingly, they introduced the *Ongoing market model*, a model of a repeating market incorporating update dynamics, and they analyzed the performance of an asynchronous tatonnement in this market. The market also incorporated warehouses (buffers) to cope with supply and demand imbalances.

Cheung, Cole and Devanur [13] showed that tatonnement is equivalent to coordinate descent on a convex function for several classes of Fisher markets, and consequently that a suitable synchronous tatonnement converges toward the market equilibrium in three general classes of markets: complementary-CES Fisher markets[3], substitute-CES Fisher markets, and Leontief Fisher markets; Cheung [11] extended this to all nested-CES Fisher markets. In this paper, we show that this equivalence enables us to perform an amortized analysis to show that the corresponding asynchronous version of tatonnement converges toward the market equilibrium in these classes of markets; indeed, our analysis also covers Fisher markets in which some buyers have substitute-CES utility functions and others have complementary ones. We also note that the tatonnement for Leontief Fisher markets analyzed in [13] had an unnatural constraint on the step sizes; our analysis removes that constraint.

Finally, we remark that it is by no means obvious that the existence of a convergence result for synchronous updating implies an analogous result for asynchronous updating. An example of a setting where an asynchronous result has yet to be achieved is proportional response dynamics [41, 4, 15].

[3] i.e., markets in which the buyers all have complementary CES utilities.

Technique of Analysis, and Comparison with the Companion Paper [16]. In a companion paper [16], we analyzed several versions of asynchronous coordinate descent. The analyses in both papers follow a common framework. We use an amortized analysis which relates the actual progress to the desired progress, where the desired progress is a constant fraction of the progress achieved with synchronous updating. The amortization is used to hide the difference between these two measures of progress by amortizing it over multiple updates. As we shall see, this difference is bounded by the squares of appropriate excess demand (resp. gradient) differences, and using Lipschitz gradient parameters, these can in turn be bounded by sum of the squares of recent changes to the prices (resp. coordinates). The final ingredient is to show that the progress is an upper bound on the square of the change to the updated price. Combining these ingredients yields a lower bound on the rate of progress.

In [16], it was assumed that the underlying convex function has some *global* finite Lipschitz gradient parameters, which is a common assumption in optimization and machine learning. The main focus there is on the maximum possible degree of parallelism which permits linear speedup, and on a number of challenges to devising rigorous and complete analyses which handle the subtle interplay between randomness (choices of coordinates) and asynchrony.

However, in the asynchronous tatonnement setting we analyze here, there are no global finite Lipschitz gradient parameters. Instead, we use *local* Lipschitz gradient parameters, as was done implicitly in [13]; the consequence is that the rate of convergence depends on the starting point. Also, the only acceptable degree of parallelism is the maximal one, i.e., all sellers are adjusting prices independently in parallel. The challenge is to devise an asynchronous analysis while keeping the price update rule reasonable, i.e., having the step size be an absolute constant which is independent of the number of goods. This calls for a somewhat different potential function and analysis from the one used for the asynchronous coordinate descent analysis in [16]; the analysis also differs quite substantially from the synchronous tatonnement analyses in [13].

Relevance to Theoretical Computer Science. Iterative procedures and dynamical systems are pervasive across multiple disciplines; a non-exhaustive list of such systems which have interested theorists includes bandwidth sharing (e.g., proportional response [39, 41, 15]), SDD linear system solvers [28, 29], distributed load balancing [26, 3], bird flocking [6], influence systems [7] and the spread of information memes across the Internet [31].

There have been many analyses of these systems, but one issue that has received relatively little attention is the timing of agents' actions. In most prior analyses, amenable timing schemes (e.g., synchronous or round robin updates) and perfect information retrieval were assumed, perhaps because they were more readily analyzed. However, typically these assumptions are unrealistic, and to better understand how these systems really behave, it is important to obtain asynchronous analyses of such systems. We believe the insight from our amortization framework may be useful in obtaining such analyses.

Other Related Work. In a similar spirit to our analysis, Cheung, Cole and Rastogi [14] analyzed asynchronous tatonnement in certain Fisher markets. This earlier work employed a potential function which drops continuously when there is no update and does not increase when an update is made. This approach could be followed for the current market setting, but in the current work, we instead use a discrete analysis which has more in common with our asynchronous coordinate descent analyses in [16]. Our work differs from [14] in two aspects. First, the update rule in [14] is more restricted: they use *average excess demand* for updates, while our update rule allows an *arbitrary value between the maximum and minimum excess* *demands*. Second, while the high-level idea is similar, our potential function is substantially different from (and more sophisticated than) the one in [14], and the classes of markets covered by the two analyses are quite different.

In a recent work, Dvijotham et al. [25] study a different asynchronous dynamics. In their setting sellers are boundedly rational and buyers are myopic (i.e., best responding). More specifically, the base (zero) level for the sellers is to be best responding, and level $k+1$ is obtained by best responding to level k sellers. They show that this system converges linearly to the market equilibrium in suitable Fisher markets including substitute-CES markets.

For the closely related topic of learning dynamics in games, where updates are based on the payoffs received by agents, again, the classical approach assumes synchronous or round-robin updates with up-to-date payoffs; models with stochastic update schedules were also studied previously (e.g., in [5, 1, 32]), while learning dynamics with delayed payoffs [34] were studied recently.

2 Preliminaries and Results

Fisher Market. In a Fisher market, there are n perfectly divisible goods and m buyers. Without loss of generality, the supply of each good is normalized to be one unit. Each buyer i has a utility function $u_i : \mathbb{R}^n_+ \to \mathbb{R}$, and a budget of size e_i. At any given price vector $\mathbf{p} \in \mathbb{R}^n_+$, each buyer purchases a maximum utility affordable collection of goods. More precisely, $\mathbf{x}_i \in \mathbb{R}^n_+$ is said to be a demand of buyer i if $\mathbf{x}_i \in \arg\max_{\mathbf{x}':\ \mathbf{x}'\cdot\mathbf{p}\leq e_i} u_i(\mathbf{x}')$.

A price vector $\mathbf{p}^* \in \mathbb{R}^n_+$ is called a *market equilibrium* if at $\mathbf{p}^*$, there exists a demand $\mathbf{x}_i$ of each buyer i such that

$$p^*_j > 0 \ \Rightarrow\ \sum_{i=1}^m x_{ij} \ = \ 1 \qquad\qquad \text{and} \qquad\qquad p^*_j = 0 \ \Rightarrow\ \sum_{i=1}^m x_{ij} \ \leq \ 1.$$

We note that in the markets we studied here, the demand at any price vector is unique. In these markets, we let $z_j := \sum_{i=1}^m x_{ij} - 1$ denote the excess demand for good j.

CES utilities. In this paper, each buyer i's utility function is of the form

$$u_i(\mathbf{x}_i) \ = \ \left(\sum_{j=1}^n a_{ij} \cdot (x_{ij})^{\rho_i} \right)^{1/\rho_i},$$

for some $-\infty \leq \rho_i < 1$, where each a_{ij} is a non-negative number. $u_i(\mathbf{x}_i)$ is called a Constant Elasticity of Substitution (CES) utility function. They are a class of utility functions often used in economic analysis. The limit as $\rho_i \to -\infty$ is called a Leontief utility, usually written as $u_i(\mathbf{x}_i) = \min_j \frac{x_{ij}}{c_{ij}}$ [4]; and the limit as $\rho_i \to 0$ is called a Cobb-Douglas utility, usually written as $\prod_j {x_{ij}}^{a_{ij}}$, with $\sum_j a_{ij} = 1$. The utilities with $\rho_i \leq 0$ capture goods that are complements, and those with $\rho_i \geq 0$ goods that are substitutes. Accordingly, when $\rho_i \leq 0$, we say the utility function is a complementary CES utility function, and when $\rho_i \geq 0$ we say it is a substitute CES utility function.

[4] The utility function $u_i(\mathbf{x}) = \min_j \frac{x_{ij}}{c_{ij}}$ can be seen as the limit of $u_i(\mathbf{x}) = \left(\sum_j \left(\frac{x_{ij}}{c_{ij}}\right)^{\rho_i} \right)^{\frac{1}{\rho_i}}$ as $\rho_i \searrow -\infty$.

Directly Related Prior Results and Our Results. Cheung, Cole and Devanur [13] showed that tatonnement is equivalent to coordinate descent on a convex function ϕ for Fisher markets with buyers having complementary-CES or Leontief utility functions (and in a later version of the paper, substitute-CES utility functions too). To be specific, [13] showed that for the convex function

$$\phi(\mathbf{p}) = \sum_{k=1}^{n} p_k \; + \; \sum_{i=1}^{m} e_i \cdot \log \hat{u}_i(\mathbf{p}),$$

where $\hat{u}_i(\mathbf{p})$ is the optimal utility that buyer i attains at price vector $\mathbf{p}$ with a unit of spending, we have that $\nabla_j \phi(\mathbf{p}) = -z_j(\mathbf{p})$. The corresponding update rule is

$$p'_j \quad \leftarrow \quad p_j \cdot [\, 1 + \lambda \cdot \min\{z_j, 1\} \,] \,, \tag{1}$$

where $\lambda > 0$ is a suitable constant. As the update rule is multiplicative, they assumed that the initial prices were positive.

As argued in [19], when the economic activity is occurring over time, it is natural to base each price update for a good on the excess demand observed by its seller since the time of the last price update to her good (possibly weighted toward more recent sales). This perceived excess demand can be written as the product of the length of the time interval with an instantaneous excess demand at some specific time in this interval, which yields the following modification of update rule (1).

$$p'_j \quad \leftarrow \quad p_j \cdot [\, 1 + \lambda \cdot \min\{\tilde{z}_j, 1\} \cdot (t - \alpha_j(t)) \,] \,, \tag{2}$$

where $\alpha_j(t)$ denotes the time of the latest update to price j *strictly* before time t, $\tilde{z}_j$ is a value between the minimum and maximum instantaneous excess demands during the time interval $(\alpha_j(t), t)$, and $\lambda > 0$ is a suitable constant. We assume that $t - \alpha_j(t) \leq 1$ for all $t \geq 0$ and for all goods j.

As we will see, having $\lambda \leq 1/25.5$ suffices. In comparison, in the synchronous version [13], $\lambda \leq 1/6$ suffices. This implies that the step sizes of the asynchronous tatonnement can be kept at a constant fraction of those used in its synchronous counterpart.

▶ **Theorem 1.** *For $\lambda \leq 1/25.5$, asynchronous tatonnement price updates using rule* (2) *converge linearly toward the market equilibrium in any complementary-CES market, and they converge in any Leontief Fisher market.*

▶ **Theorem 2.** *Let $\mathcal{M}$ be a Fisher market in which buyers have CES utility functions. Suppose that $\rho := \max_i \rho_i < 1$ and $\min_i \rho_i > -\infty$ in $\mathcal{M}$. Let $E := \max\{1/(1-\rho)\,,\ 1\}$. Then for $\lambda \leq 1/(26E)$, asynchronous tatonnement price updates using rule* (2) *converge linearly toward the market equilibrium.*

Here, we focus on the result concerning complementary-CES Fisher markets; the analyses for the other cases are deferred to the full version. The analysis for Theorem 2 is just a small modification of the complementary case. For the Leontief Fisher markets, while the first part of the analysis is identical to the complementary case, this is not enough to demonstrate convergence, and to do so requires substantially more effort.

In an earlier version of this paper [12] on arXiv, we proved Theorem 1 (except that λ was slightly larger) using a potential function which decreases continuously over time, as was the case for the analyses in [20, 14] also. We believe the current analysis is considerably simpler. The main advantage of the prior analysis at this point is that we extended it to account for the warehouses in the Ongoing market model, albeit with a quite non-trivial argument. This seems possible with the potential function in the present paper too, but we suspect it would be of interest to at most a few specialists.

Standard Notation in Coordinate Descent. Let $\mathbf{e}_j$ denote the unit vector along coordinate (in our context, price) j. A function F is L-Lipschitz-smooth if for any $\mathbf{p}, \Delta\mathbf{p} \in \mathbb{R}^n$, $\|\nabla F(\mathbf{p}+\Delta\mathbf{p}) - \nabla F(\mathbf{p})\| \leq L \cdot \|\Delta\mathbf{p}\|$. For any coordinates j, k, a function F is L_{jk}-Lipschitz-smooth if for any $\mathbf{p} \in \mathbb{R}^n$ and $r \in \mathbb{R}$, $|\nabla_k F(\mathbf{p} + r \cdot \mathbf{e}_j) - \nabla_k F(\mathbf{p})| \leq L_{jk} \cdot |r|$. Also, as is standard, L_j denotes L_{jj}.

3 Key Ideas and Lemmas

For simplicity, we assume that at any particular time t, there is at most one update to one good. In general, since there is no coordination between price updates of different goods, it is possible that the prices of two goods are updated at the same *moment*; but by using any arbitrary tie-breaking (perturbation) rule, our analysis extends to such cases.

Recall update rule (2). For the purposes of our analysis, for each update we now need to know the elapsed time since the previous update to the same coordinate, or since time 0 if it is the first update to that coordinate; for the update at time τ, we denote this by Δt_τ. As explained in [19], in the Ongoing market model, all the sellers need to know is the size of their warehouse stock at the times of the current update and the previous update, which seem to be very natural information. We let $\alpha_j(\tau)$ denote the time of the most recent update to p_j *strictly* before time τ, or time 0 if there is no previous update to this price. We let $\alpha(\tau)$ denote the time of the most recent update to *any* price *strictly* before time τ, or time 0 if there is no previous update to any price. And we let $\delta_\tau := \tau - \alpha(\tau)$, the elapsed time since the most recent previous update to any price.

Suppose there is an update at time t. We let p_{k_t} denote the price updated at time t, we let $p_{k_t}^t$ denote its updated value, and $p_{k_t}^{t-}$ its value right before this update; note that $p_{k_t}^{t-} \equiv p_{k_t}^{\alpha(t)} \equiv p_{k_t}^{t-\delta_t}$. Let $\Delta p_{k_t}^t := p_{k_t}^t - p_{k_t}^{t-}$. Also, we let $\tilde{z}_{k_t}^t$ denote the value of the excess demand used in the update to price p_{k_t} at time t. Finally, we let $\Gamma_{k_t}^t := \max\{1, \tilde{z}_{k_t}^t\}/(\lambda p_{k_t}^{t-})$. Then update rule (2) can be rewritten in the following form:

$$p_{k_t}^t \quad \leftarrow \quad p_{k_t}^{t-} \; + \; \frac{1}{\Gamma_{k_t}^t} \cdot \tilde{z}_{k_t}^t \cdot \Delta t. \tag{3}$$

In our analysis, when we write $\sum_{\tau \in I}$, where I is a time interval, the summation is summing over all updates that occurred in time interval I.

Let z_k^t be the instantaneous excess demand for good k right before a price update at time t. We note that $z_k^t = -\nabla_k \phi(\mathbf{p}^{\alpha(t)})$. For each update τ, let $z_k^{\max,\tau}$ and $z_k^{\min,\tau}$ denote the maximum and minimum of accurate excess demand values of good k in the time interval $(\alpha_{k_\tau}(\tau), \tau)$.

We also need to define local Lipschitz parameters: $L_{jk}^{[\tau_a,\tau_b]}$ is an upper bound on the Lipschitz gradient parameter L_{jk} of the function ϕ within a rectangular hull of those prices which might appear in the time interval $[\tau_a, \tau_b]$. Observe that in update rule (2), since $|\min\{\tilde{z}_j, 1\}| \leq 1$ always, the above-mentioned rectangular hull is finitely bounded, and furthermore, it shrinks as λ gets smaller.

We use the following three lemmas. Lemma 3 is modified from a standard lemma in coordinate descent to accommodate local Lipschitz gradient continuity. Lemma 4 is a direct consequence of the Power-Mean inequality. Lemma 5 is a simple algebra exercise. See the full version for the missing proofs.

▶ **Lemma 3.** *Suppose there is an update to coordinate k_t at time t according to rule (2), and suppose that $\lambda \leq 1/10$. Recall that $\tilde{z}_{k_t}$ is the value used in applying rule (2). Then*

$$\phi(\mathbf{p}^{\alpha(t)}) - \phi(\mathbf{p}^t) \quad \geq \quad \frac{\Gamma^t_{k_t}}{4} \cdot \frac{(\Delta p^t_{k_t})^2}{\Delta t} \; - \; \frac{1}{\Gamma^t_{k_t}} \cdot \left(z^t_{k_t} - \tilde{z}_{k_t}\right)^2 \cdot |\Delta t|.$$

▶ **Lemma 4.** *Suppose that $w_1, w_2, \cdots, w_\ell$ and $y_1, y_2, \cdots, y_\ell$ are non-negative numbers. Then* $\left(\sum_{j=1}^{\ell} w_j y_j\right)^2 \quad \leq \quad \left(\sum_{j=1}^{\ell} w_j\right)\left(\sum_{j=1}^{\ell} w_j \cdot (y_j)^2\right)$.

▶ **Lemma 5.** *In Lemma 3, suppose that $\tilde{z}'_{k_t}$ were used instead of $\tilde{z}_{k_t}$ in update rule (3), but with $\Gamma^t_{k_t}$ unchanged. Let the new Δp_{k_t} value be $\Delta' p^t_{k_t}$. Then*

$$\Gamma^t_{k_t} \cdot \frac{(\Delta p^t_{k_t})^2}{\Delta t} \quad \geq \quad \frac{\Gamma^t_{k_t}}{2} \cdot \frac{(\Delta' p^t_{k_t})^2}{\Delta t} \; - \; \frac{1}{\Gamma^t_{k_t}} \cdot (\tilde{z}_{k_t} - \tilde{z}'_{k_t})^2 \cdot \Delta t.$$

In the RHS of the inequality in Lemma 3, we call the first term, $\frac{\Gamma^t_{k_t}}{4} \cdot \frac{(\Delta p^t_{k_t})^2}{\Delta t}$, a progress term, and we call the second term, $-\frac{1}{\Gamma^t_{k_t}} \cdot \left(z^t_{k_t} - \tilde{z}_{k_t}\right)^2 \cdot |\Delta t|$, an error term. The progress term is cut into two halves. The first half will be used to demonstrate progress of the convergence, while the second half will be saved to compensate for the error terms in subsequent updates. Accordingly, we design a potential function $\Phi(t)$ of the form $\Phi(t) := \phi(\mathbf{p}^t) + A(t)$, where $A(t) \geq 0$ for all t and $A(0) = 0$. We call $A(t)$ the *amortization bank*; its purpose is to save portions of the progress terms for future compensations.

By showing that $\Phi(t)$ reduces by a constant fraction ε in every $\mathcal{O}(1)$ time units, we can deduce that $\phi(\mathbf{p}^t) \leq \Phi(t) \leq \Phi(0) \cdot (1-\varepsilon)^{\Theta(t)} = \phi(\mathbf{p}^\circ) \cdot (1-\varepsilon)^{\Theta(t)}$, as desired.

We define the function $A(t)$ as follows:

$$A(t) = c_1 \sum_{\tau \in (t-1,t]} \; \sum_{k \neq k_\tau} \left(1 + \mathbb{1}[\mathcal{E}(k,\tau,t)]\right) \cdot L^{[\tau,\tau+1]}_{k_\tau,k} \cdot \frac{p^\tau_k}{p^{\tau-}_{k_\tau}} \cdot \frac{\left(\Delta p^\tau_{k_\tau}\right)^2}{\Delta t_\tau},$$

where $c_1 > 0$ is a constant we will determine later, and $\mathcal{E}(k,\tau,t)$ denotes the event that price k is *not* updated during the time interval $(\tau, t]$.

4 Analysis

Suppose there is an update at time $t \geq 2$. We let t_a denote the time of the latest update *strictly* before time $(t-2)$, if any; otherwise, we let $t_a = 0$. We let t_b denote the time of the earliest update in the time interval $[t-1, t]$. By Lemma 3,

$$\begin{aligned}
&\sum_{\tau \in (t_a,t]} \left[\Phi(\alpha(\tau)) - \Phi(\tau)\right] \\
&\geq \sum_{\tau \in (t_a,t]} \left[\frac{\Gamma^\tau_{k_\tau}}{4} \cdot \frac{(\Delta p^\tau_{k_\tau})^2}{\Delta t_\tau} - \frac{1}{\Gamma^\tau_{k_\tau}} (z^\tau_{k_\tau} - \tilde{z}^\tau_{k_\tau})^2 \cdot \Delta t_\tau \right] + A(t_a) - A(t) \\
&\overset{(*)}{\geq} \sum_{\tau \in (t_a,t]} \left[\frac{\Gamma^\tau_{k_\tau}}{4} \cdot \frac{(\Delta p^\tau_{k_\tau})^2}{\Delta t_\tau} - \frac{1}{\Gamma^\tau_{k_\tau}} (z^{\max,\tau}_{k_\tau} - z^{\min,\tau}_{k_\tau})^2 \cdot \Delta t_\tau \right] + A(t_a) - A(t). \qquad (4)
\end{aligned}$$

Inequality $(*)$ holds because in the tatonnement setting, both the accurate excess demand $z^\tau_{k_\tau}$ and the inaccurate excess demand $\tilde{z}^\tau_{k_\tau}$ must lie between $z^{\max,\tau}_{k_\tau}$ and $z^{\min,\tau}_{k_\tau}$.

For any $\nu \in (\alpha_{k_\tau}(\tau), \tau)$, let $\Delta' p^\nu_{k_\tau}$ denote the Δp_{k_τ} value if there were an update at time ν to price p_{k_τ} using the accurate excess demand $z^\nu_{k_\tau}$. By Lemma 5, for each $\tau \in (t_a, t]$,

$$\begin{aligned}
&\frac{\Gamma^\tau_{k_\tau}}{8} \cdot \frac{(\Delta p^\tau_{k_\tau})^2}{\Delta t_\tau} \\
&\geq \sum_{\nu \in (\max\{t_a, \alpha_{k_\tau}(\tau)\}, \tau]} \left[\frac{\Gamma^\tau_{k_\tau}}{16} \cdot \frac{(\Delta' p^\nu_{k_\tau})^2}{\Delta t_\tau} \cdot \frac{\delta_\nu}{\Delta t_\tau} - \frac{1}{8\Gamma^\tau_{k_\tau}} (z^\nu_{k_\tau} - \tilde{z}^\tau_{k_\tau})^2 \cdot \delta_\nu \right] \\
&\geq \frac{\Gamma^\tau_{k_\tau}}{16} \left(\sum_{\nu \in (\max\{t_a, \alpha_{k_\tau}(\tau)\}, \tau]} \frac{(\Delta' p^\nu_{k_\tau})^2}{(\Delta t_\tau)^2} \cdot \delta_\nu \right) - \frac{1}{8\Gamma^\tau_{k_\tau}} \cdot (z^{\max,\tau}_{k_\tau} - z^{\min,\tau}_{k_\tau})^2. \qquad (5)
\end{aligned}$$

For any k and any time ν, let $\beta_k(\nu)$ denote the time of the earliest update to price k on or after time ν. Combining (4) and (5) yields

$$\begin{aligned}
&\sum_{\tau \in (t_a, t]} [\Phi(\alpha(\tau)) - \Phi(\tau)] \\
&\geq \sum_{\tau \in (t_a, t]} \sum_{\nu \in (\max\{t_a, \alpha_{k_\tau}(\tau)\}, \tau]} \frac{\Gamma^\tau_{k_\tau}}{16} \cdot \frac{(\Delta' p^\nu_{k_\tau})^2}{(\Delta t_\tau)^2} \cdot \delta_\nu \\
&\quad + \left[\sum_{\tau \in (t_a, t]} \frac{\Gamma^\tau_{k_\tau}}{8} \cdot \frac{(\Delta p^\tau_{k_\tau})^2}{\Delta t_\tau} + A(t_a) - A(t) \right] - \sum_{\tau \in (t_a, t]} \frac{9}{8\Gamma^\tau_{k_\tau}} \cdot (z^{\max,\tau}_{k_\tau} - z^{\min,\tau}_{k_\tau})^2 \\
&\geq \frac{1}{16} \sum_{\nu \in (t_a, t_b]} \delta_\nu \sum_{k=1}^n \frac{\Gamma_k^{\beta_k(\nu)} \cdot (\Delta' p^\nu_k)^2}{(\Delta t_{\beta_k(\nu)})^2} - \sum_{\tau \in (t_a, t]} \frac{9}{8\Gamma^\tau_{k_\tau}} \cdot (z^{\max,\tau}_{k_\tau} - z^{\min,\tau}_{k_\tau})^2 \\
&\quad + \left[\sum_{\tau \in (t_a, t]} \frac{\Gamma^\tau_{k_\tau}}{8} \cdot \frac{(\Delta p^\tau_{k_\tau})^2}{\Delta t_\tau} + A(t_a) - A(t) \right] \\
&\geq \frac{1}{16} \sum_{\nu \in (t_a, t_b]} \delta_\nu \left[\sum_{k=1}^n \frac{\Gamma_k^{\beta_k(\nu)} (\Delta' p^\nu_k)^2}{(\Delta t_{\beta_k(\nu)})^2} + c_2 \cdot A(\alpha(\nu)) \right] - \sum_{\tau \in (t_a, t]} \frac{9}{8\Gamma^\tau_{k_\tau}} \cdot (z^{\max,\tau}_{k_\tau} - z^{\min,\tau}_{k_\tau})^2 \\
&\quad + \left[\sum_{\tau \in (t_a, t]} \frac{\Gamma^\tau_{k_\tau}}{8} \cdot \frac{(\Delta p^\tau_{k_\tau})^2}{\Delta t_\tau} + A(t_a) - A(t) - \frac{c_2}{16} \sum_{\nu \in (t_a, t_b]} \delta_\nu \cdot A(\alpha(\nu)) \right],
\end{aligned}$$

for some small constant $c_2 > 0$ we will determine later.

In [13], it was proved that the function ϕ is strongly convex in any region bounded away from zero prices, and that the maximum Γ value throughout the tatonnement is upper bounded by a finite constant which depends on the starting price $\mathbf{p}^\circ$.[5] We denote the finite upper bound on all Γ's by $\overline{\Gamma}$, and the strong convexity parameter of ϕ by μ_ϕ, which also depends on the starting prices. We let $\varepsilon := \mu_\phi / \overline{\Gamma}$. Then it is a standard fact in optimization that

$$\sum_{k=1}^n \frac{\Gamma_k^{\beta_k(\nu)} (\Delta' p^\nu_k)^2}{(\Delta t_{\beta_k(\nu)})^2} = \sum_{k=1}^n \frac{1}{\Gamma_k^{\beta_k(\nu)}} \cdot (z^\nu_k)^2 \geq \sum_{k=1}^n \frac{1}{\overline{\Gamma}} \cdot (z^\nu_k)^2 \geq \varepsilon \cdot \phi(\mathbf{p}^{\alpha(\nu)}).$$

[5] Their argument concerned the synchronous setting, but it can be reused without change for the asynchronous setting.

Setting $\varepsilon' = \min\{\varepsilon, c_2\}/16$ yields

$$\begin{aligned}
&\sum_{\tau\in(t_a,t]} [\Phi(\alpha(\tau)) - \Phi(\tau)] \\
&\geq \ \varepsilon' \sum_{\nu\in(t_a,t_b]} \delta_\nu \cdot \Phi(\alpha(\nu)) \ - \sum_{\tau\in(t_a,t]} \frac{9}{8\Gamma^\tau_{k_\tau}} \cdot (z^{\max,\tau}_{k_\tau} - z^{\min,\tau}_{k_\tau})^2 \\
&\quad + \left[\sum_{\tau\in(t_a,t]} \frac{\Gamma^\tau_{k_\tau}}{8} \cdot \frac{(\Delta p^\tau_{k_\tau})^2}{\Delta t_\tau} \ + \ A(t_a) \ - \ A(t) \ - \ \frac{c_2}{16} \sum_{\nu\in(t_a,t_b]} \delta_\nu \cdot A(\alpha(\nu)) \right]. \qquad (6)
\end{aligned}$$

In the subsections below, we will prove that for a suitable choice of the Γ parameters and c_1, c_2, the final two terms of (6), in sum, are non-negative. Also, we will show that Φ is decreasing over time. With these, the above inequality implies that

$$\Phi(t_a) - \Phi(t) \ = \sum_{\tau\in(t_a,t]} [\Phi(\alpha(\tau)) - \Phi(\tau)] \ \geq \ \varepsilon' \sum_{\nu\in(t_a,t_b]} \delta_\nu \cdot \Phi(\alpha(\nu)) \ \geq \ \varepsilon' \cdot \Phi(t),$$

and hence $\Phi(t) \ \leq \ \Phi(t_a)/(1+\varepsilon')$. By iterating this (note that $t > t_a \geq t-3$), we obtain

$$\phi(\mathbf{p}^t) \ \leq \ \Phi(t) \ \leq \ (1+\varepsilon')^{-(t/3-1)} \cdot \Phi(0) \ = \ (1+\varepsilon')^{-(t/3-1)} \cdot \phi(\mathbf{p}^\circ),$$

thus demonstrating linear convergence.

4.1 Φ is a Decreasing Function

For any time τ at which there is an update, by Lemma 3 and the definition of A, we have

$$\begin{aligned}
&\Phi(\alpha(\tau)) - \Phi(\tau) \\
&\geq \frac{\Gamma^\tau_{k_\tau}}{4} \cdot \frac{(\Delta p^\tau_{k_\tau})^2}{\Delta t_\tau} \ - \ \frac{1}{\Gamma^\tau_{k_\tau}} (z^\tau_{k_\tau} - \tilde{z}^\tau_{k_\tau})^2 \cdot \Delta t_\tau \ + \ c_1 \sum_{\nu\in(\alpha_{k_\tau}(\tau),\tau)} L^{[\nu,\nu+1]}_{k_\nu k_\tau} \cdot \frac{p^{\tau-}_{k_\tau}}{p^{\nu-}_{k_\nu}} \cdot \frac{\left(\Delta p^\nu_{k_\nu}\right)^2}{\Delta t_\nu} \\
&\quad - \ 2c_1 \sum_{k\neq k_\tau} L^{[\tau,\tau+1]}_{k_\tau,k} \cdot \frac{p^\tau_k}{p^{\tau-}_{k_\tau}} \cdot \frac{\left(\Delta p^\tau_{k_\tau}\right)^2}{\Delta t_\tau}.
\end{aligned}$$

Next,

$$\begin{aligned}
&\left(z^{\max,\tau}_{k_\tau} - z^{\min,\tau}_{k_\tau}\right)^2 \\
&\leq \left(\sum_{\nu\in(\alpha_{k_\tau}(\tau),\tau)} L^{[\nu,\tau]}_{k_\nu,k_\tau} \left|\Delta p^\nu_{k_\nu}\right| \right)^2 \\
&= \left(\sum_{\nu\in(\alpha_{k_\tau}(\tau),\tau)} \left(L^{[\nu,\tau]}_{k_\nu,k_\tau} \cdot \Delta t_\nu \cdot \frac{p^{\nu-}_{k_\nu}}{p^{\nu-}_{k_\tau}} \right) \cdot \left(\frac{\left|\Delta p^\nu_{k_\nu}\right|}{\Delta t_\nu} \cdot \frac{p^{\nu-}_{k_\tau}}{p^{\nu-}_{k_\nu}} \right) \right)^2 \\
&\leq \left(\sum_{\nu\in(\alpha_{k_\tau}(\tau),\tau)} L^{[\nu,\tau]}_{k_\nu,k_\tau} \cdot \Delta t_\nu \cdot \frac{p^{\nu-}_{k_\nu}}{p^{\nu-}_{k_\tau}} \right) \left(\sum_{\nu\in(\alpha_{k_\tau}(\tau),\tau)} L^{[\nu,\tau]}_{k_\nu,k_\tau} \cdot \frac{p^{\nu-}_{k_\tau}}{p^{\nu-}_{k_\nu}} \cdot \frac{\left(\Delta p^\nu_{k_\nu}\right)^2}{\Delta t_\nu} \right) \\
&\hfill \text{(by Lemma 4)} \\
&\leq \left(\sum_{\nu\in(\alpha_{k_\tau}(\tau),\tau)} L^{[\alpha_{k_\tau}(\tau),\tau]}_{k_\nu,k_\tau} \cdot \Delta t_\nu \cdot \frac{p^{\nu-}_{k_\nu}}{p^{\tau-}_{k_\tau}} \right) \left(\sum_{\nu\in(\alpha_{k_\tau}(\tau),\tau)} L^{[\nu,\nu+1]}_{k_\nu,k_\tau} \cdot \frac{p^{\tau-}_{k_\tau}}{p^{\nu-}_{k_\nu}} \cdot \frac{\left(\Delta p^\nu_{k_\nu}\right)^2}{\Delta t_\nu} \right). \qquad (7)
\end{aligned}$$

Combining the above two equations and recalling that $\Delta t_\tau \leq 1$ yields

$$\begin{aligned}
&\Phi(\alpha(\tau)) - \Phi(\tau) \\
&\geq \left(\frac{\Gamma^\tau_{k_\tau}}{4} - 2c_1 \sum_{k\neq k_\tau} L^{[\tau,\tau+1]}_{k_\tau,k} \cdot \frac{p^\tau_k}{p^{\tau-}_{k_\tau}}\right) \frac{\left(\Delta p^\tau_{k_\tau}\right)^2}{\Delta t_\tau} \\
&\qquad + \left(c_1 - \frac{1}{\Gamma^\tau_{k_\tau}} \sum_{\nu\in(\alpha_{k_\tau}(\tau),\tau)} L^{[\alpha_{k_\tau}(\tau),\tau]}_{k_\nu,k_\tau} \cdot \Delta t_\nu \cdot \frac{p^{\nu-}_{k_\nu}}{p^{\tau-}_{k_\tau}}\right) \\
&\qquad\qquad \cdot \left(\sum_{\nu\in(\alpha_{k_\tau}(\tau),\tau)} L^{[\nu,\nu+1]}_{k_\nu,k_\tau} \cdot \frac{p^{\tau-}_{k_\tau}}{p^{\nu-}_{k_\nu}} \cdot \frac{\left(\Delta p^\nu_{k_\nu}\right)^2}{\Delta t_\nu}\right). \qquad (8)
\end{aligned}$$

Thus, for Φ to be decreasing, we impose the following conditions (the second one is stronger than what is needed at this point):

$$\Gamma^\tau_{k_\tau} \ \geq\ 8c_1 \sum_{k\neq k_\tau} L^{[\tau,\tau+1]}_{k_\tau,k} \cdot \frac{p^\tau_k}{p^{\tau-}_{k_\tau}} \quad \text{and} \quad \Gamma^\tau_{k_\tau} \ \geq\ \frac{2}{c_1} \sum_{\nu\in(\alpha_{k_\tau}(\tau),\tau)} L^{[\alpha_{k_\tau}(\tau),\tau]}_{k_\nu,k_\tau} \cdot \Delta t_\nu \cdot \frac{p^{\nu-}_{k_\nu}}{p^{\tau-}_{k_\tau}}. \qquad (9)$$

4.2 The Sum of the Last Two Terms in (6) is Non-negative

It remains to show that the sum of the last two terms in (6) is non-negative, i.e.,

$$\begin{aligned}
&\sum_{\tau\in(t_a,t]} \frac{\Gamma^\tau_{k_\tau}}{8} \cdot \frac{(\Delta p^\tau_{k_\tau})^2}{\Delta t_\tau} - \frac{c_2}{16} \sum_{\nu\in(t_a,t_b]} \delta_\nu \cdot A(\alpha(\nu)) \ +\ A(t_a) - A(t) \\
&\qquad\qquad \geq\ \sum_{\tau\in(t_a,t]} \frac{9}{8\Gamma^\tau_{k_\tau}} \cdot (z^{\max,\tau}_{k_\tau} - z^{\min,\tau}_{k_\tau})^2. \qquad (10)
\end{aligned}$$

We first simplify the LHS using the definition of A:

$$\begin{aligned}
&\sum_{\tau\in(t_a,t]} \frac{\Gamma^\tau_{k_\tau}}{8} \cdot \frac{(\Delta p^\tau_{k_\tau})^2}{\Delta t_\tau} \ -\ \frac{c_2}{16} \sum_{\nu\in(t_a,t_b]} \delta_\nu \cdot A(\alpha(\nu)) \ +\ A(t_a) \ -\ A(t) \\
&\geq \sum_{\tau\in(t_a,t]} \frac{\Gamma^\tau_{k_\tau}}{8} \cdot \frac{(\Delta p^\tau_{k_\tau})^2}{\Delta t_\tau} \ -\ \frac{c_2}{16} \sum_{\nu\in(t_a-1,t_b]} 4c_1 \sum_{k\neq k_\nu} L^{[\nu,\nu+1]}_{k_\nu,k} \cdot \frac{p^\nu_k}{p^{\nu-}_{k_\nu}} \cdot \frac{\left(\Delta p^\nu_{k_\nu}\right)^2}{\Delta t_\nu} \\
&\qquad\qquad + \ c_1 \sum_{\nu\in(t_a-1,t_a]} \sum_{k\neq k_\nu} L^{[\nu,\nu+1]}_{k_\nu,k} \cdot \frac{p^\nu_k}{p^{\nu-}_{k_\nu}} \cdot \frac{\left(\Delta p^\nu_{k_\nu}\right)^2}{\Delta t_\nu} \\
&\qquad\qquad - \ 2c_1 \sum_{\nu\in(t-1,t]} \sum_{k\neq k_\nu} L^{[\nu,\nu+1]}_{k_\nu,k} \cdot \frac{p^\nu_k}{p^{\nu-}_{k_\nu}} \cdot \frac{\left(\Delta p^\nu_{k_\nu}\right)^2}{\Delta t_\nu} \\
&\geq \left(c_1 - \frac{c_1c_2}{4}\right) \sum_{\nu\in(t_a-1,t_a]} \sum_{k\neq k_\nu} L^{[\nu,\nu+1]}_{k_\nu,k} \cdot \frac{p^\nu_k}{p^{\nu-}_{k_\nu}} \cdot \frac{\left(\Delta p^\nu_{k_\nu}\right)^2}{\Delta t_\nu} \\
&\qquad + \sum_{\tau\in(t_a,t]} \left[\frac{\Gamma^\tau_{k_\tau}}{8} - \left(\frac{c_1c_2}{4} + 2c_1\right) \sum_{k\neq k_\tau} L^{[\tau,\tau+1]}_{k_\tau,k} \cdot \frac{p^\tau_k}{p^{\tau-}_{k_\tau}}\right] \frac{(\Delta p^\tau_{k_\tau})^2}{\Delta t_\tau}.
\end{aligned}$$

By imposing the requirement that $\Gamma^\tau_{k_\tau} \ \geq\ 8c_3 \ \sum_{k\neq k_\tau} \ L^{[\tau,\tau+1]}_{k_\tau,k} \cdot \frac{p^\tau_k}{p^{\tau-}_{k_\tau}}$, for some constant

$c_3 > 0$ which we will determine later, we obtain

$$\sum_{\tau\in(t_a,t]} \frac{\Gamma^\tau_{k_\tau}}{8} \frac{(\Delta p^\tau_{k_\tau})^2}{\Delta t_\tau} \;-\; \frac{c_2}{16} \sum_{\nu\in(t_a,t_b]} \delta_\nu \cdot A(\alpha(\nu)) \;+\; A(t_a) \;-\; A(t)$$

$$\geq \;\; \min\left\{ c_1 - \frac{c_1c_2}{4} \; , \; c_3 - \frac{c_1c_2}{4} - 2c_1 \right\} \cdot \sum_{\tau\in(t_a-1,t]} \sum_{k\neq k_\tau} L^{[\tau,\tau+1]}_{k_\tau,k} \cdot \frac{p^\tau_k}{p^{\tau-}_{k_\tau}} \frac{(\Delta p^\tau_{k_\tau})^2}{\Delta t_\tau}.$$

On the other hand, by (7) and by the second condition imposed in (9),

$$\begin{aligned}
\sum_{\tau\in(t_a,t]} \frac{9}{8\Gamma^\tau_{k_\tau}} \cdot (z^{\max,\tau}_{k_\tau} - z^{\min,\tau}_{k_\tau})^2 \;&\leq\; \frac{9c_1}{16} \sum_{\tau\in(t_a,t]} \;\; \sum_{\nu\in(\alpha_{k_\tau}(\tau),\tau)} L^{[\nu,\nu+1]}_{k_\nu,k_\tau} \cdot \frac{p^{\tau-}_{k_\tau}}{p^{\nu-}_{k_\nu}} \cdot \frac{\left(\Delta p^\nu_{k_\nu}\right)^2}{\Delta t_\nu} \\
&=\; \frac{9c_1}{16} \sum_{\tau\in(t_a,t]} \sum_{\nu\in(\alpha_{k_\tau}(\tau),\tau)} L^{[\nu,\nu+1]}_{k_\nu,k_\tau} \cdot \frac{p^\nu_{k_\tau}}{p^{\nu-}_{k_\nu}} \cdot \frac{\left(\Delta p^\nu_{k_\nu}\right)^2}{\Delta t_\nu} \\
&\leq\; \frac{9c_1}{16} \sum_{\nu\in(t_a-1,t]} \sum_{k\neq k_\nu} L^{[\nu,\nu+1]}_{k_\nu,k} \cdot \frac{p^\nu_k}{p^{\nu-}_{k_\nu}} \frac{(\Delta p^\nu_{k_\nu})^2}{\Delta t_\nu}.
\end{aligned}$$

By the above two inequalities, to satisfy (10), it suffices to have $\frac{9c_1}{16} \leq \min\left\{ c_1 - \frac{c_1c_2}{4} , c_3 - \frac{c_1c_2}{4} - 2c_1 \right\}$. There are multiple possible choices for c_1, c_2, c_3. We choose $c_3 = 21c_1/8$ and $c_2 = 1/4$. To summarize, we need the Γ parameters to satisfy

$$\Gamma^\tau_{k_\tau} \;\geq\; 21c_1 \sum_{k\neq k_\tau} L^{[\tau,\tau+1]}_{k_\tau,k} \cdot \frac{p^\tau_k}{p^{\tau-}_{k_\tau}} \quad\text{and}\quad \Gamma^\tau_{k_\tau} \;\geq\; \frac{2}{c_1} \sum_{\nu\in(\alpha_{k_\tau}(\tau),\tau)} L^{[\alpha_{k_\tau}(\tau),\tau]}_{k_\nu,k_\tau} \cdot \Delta t_\nu \cdot \frac{p^{\nu-}_{k_\nu}}{p^{\tau-}_{k_\tau}}. \quad (11)$$

Our remaining tasks are to derive upper bounds on the two summations in (11).

4.3 Upper Bounds on the Local Lipschitz Parameters, and Determining the Γ's

Suppose in a Fisher market with buyers having CES utility functions, each buyer i has a budget of e_i, and her CES utility function has parameter ρ_i. For each i, let $\theta_i := \rho_i/(\rho_i - 1)$. As we have discussed in Section 2, at any given price vector $\mathbf{p} \in \mathbb{R}^n_+$, buyer i computes the demand-maximizing bundle of goods costing at most e_i; we let $x_{i\ell}(\mathbf{p})$ denote buyer i's demand for good ℓ at price vector $\mathbf{p}$.

In a Fisher market with buyers having complementary-CES utility functions, the following properties are well-known. (See [14].)

1. For any $k \neq j$,

$$\left| \frac{\partial^2 \phi}{\partial p_j \, \partial p_k}(\mathbf{p}) \right| \;=\; \sum_{i=1}^m \frac{\theta_i \; x_{ij}(\mathbf{p}) \; x_{ik}(\mathbf{p})}{e_i} \;\leq\; \sum_{i=1}^m \frac{x_{ij}(\mathbf{p}) \; x_{ik}(\mathbf{p})}{e_i}.$$

2. Given positive price vector $\mathbf{p}$, for any $0 < r_1 < r_2$, let $\mathbf{p}'$ be prices such that for all ℓ, $r_1 p_\ell \leq p'_\ell \leq r_2 p_\ell$. Then for all ℓ, $\frac{1}{r_2} x_\ell(\mathbf{p}) \leq x_\ell(\mathbf{p}') \leq \frac{1}{r_1} x_\ell(\mathbf{p})$.

▶ **Lemma 6.** *If the parameter λ in update rule* (2) *is at most* $1/10$*, then*

$$\sum_{k\neq k_\tau} L^{[\tau,\tau+1]}_{k_\tau,k} \cdot \frac{p^\tau_k}{p^{\tau-}_{k_\tau}} \;\leq\; e^{4\lambda(\lambda+1)} \cdot \frac{x_{k_\tau}(\mathbf{p}^{\tau-})}{p^{\tau-}_{k_\tau}}$$

and

$$\sum_{\nu\in(\alpha_{k_\tau}(\tau),\tau)} L^{[\alpha_{k_\tau}(\tau),\tau]}_{k_\nu,k_\tau}\cdot\Delta t_\nu\cdot\frac{p^{\nu-}_{k_\nu}}{p^{\tau-}_{k_\tau}} \;\le\; 2e^{8\lambda(\lambda+1)}\cdot\frac{x_{k_\tau}(\mathbf{p}^{\tau-})}{p^{\tau-}_{k_\tau}}.$$

Proof. Let $\lambda'=2\lambda(\lambda+1)$. Since $\lambda\le 1/10$, it is easy to observe that for any $\nu\in[\tau,\tau+1]$ and for any k (including coordinate k_τ),

$$e^{-\lambda'}\cdot p^{\tau-}_k \;\le\; p^\nu_k \;\le\; e^{\lambda'}\cdot p^{\tau-}_k, \tag{12}$$

on noting that the Δt_ν terms span up to 2 time units.

Accordingly, let $\tilde{P}:=\left\{(\tilde{p}_1,\tilde{p}_2,\cdots,\tilde{p}_n)\;\middle|\;\forall k\in[n],\; e^{-\lambda'}\cdot p^{\tau-}_k\;\le\;\tilde{p}_k\;\le\;e^{\lambda'}\cdot p^{\tau-}_k\right\}$. Then,

$$\begin{aligned}
\sum_{k\ne k_\tau} L^{[\tau,\tau+1]}_{k_\tau,k}\cdot\frac{p^\tau_k}{p^{\tau-}_{k_\tau}} &\le \frac{1}{p^{\tau-}_{k_\tau}}\sum_{k\ne k_\tau}\left(\max_{\tilde{p}\in\tilde{P}}\left|\frac{\partial^2\phi}{\partial p_{k_\tau}\,\partial p_k}(\tilde{p})\right|\right)\cdot p^\tau_k\\
&\le \frac{1}{p^{\tau-}_{k_\tau}}\sum_{k\ne k_\tau}\sum_{i=1}^m\frac{(e^{\lambda'}x_{ik_\tau}(\mathbf{p}^{\tau-}))\cdot(e^{\lambda'}x_{ik}(\mathbf{p}^{\tau-}))}{e_i}\cdot p^{\tau-}_k \qquad \text{(by Properties 1 and 2)} \qquad (13)\\
&\le \frac{e^{2\lambda'}}{p^{\tau-}_{k_\tau}}\sum_{i=1}^m x_{ik_\tau}(\mathbf{p}^{\tau-})\sum_{k\ne k_\tau}\frac{x_{ik}(\mathbf{p}^{\tau-})\cdot p^{\tau-}_k}{e_i}\\
&\le \frac{e^{2\lambda'}}{p^{\tau-}_{k_\tau}}\sum_{i=1}^m x_{ik_\tau}(\mathbf{p}^{\tau-}) \qquad \text{(the second summation}\le 1\text{, due to the budget constraint)}\\
&= e^{2\lambda'}\cdot\frac{x_{k_\tau}(\mathbf{p}^{\tau-})}{p^{\tau-}_{k_\tau}}. \qquad (14)
\end{aligned}$$

For the time range $\nu\in[\alpha_{k_\tau}(\tau),\tau]$, inequality (12) also holds. Thus,

$$\begin{aligned}
&\sum_{\nu\in(\alpha_{k_\tau}(\tau),\tau)} L^{[\alpha_{k_\tau}(\tau),\tau]}_{k_\nu,k_\tau}\cdot\Delta t_\nu\cdot\frac{p^{\nu-}_{k_\nu}}{p^{\tau-}_{k_\tau}}\\
\le\;& \frac{1}{p^{\tau-}_{k_\tau}}\sum_{\nu\in(\alpha_{k_\tau}(\tau),\tau)} L^{[\alpha_{k_\tau}(\tau),\tau]}_{k_\nu,k_\tau}\cdot\Delta t_\nu\cdot e^{2\lambda'}\cdot p^{\tau-}_{k_\nu}\\
\le\;& \frac{e^{2\lambda'}}{p^{\tau-}_{k_\tau}}\sum_{k\ne k_\tau} L^{[\alpha_{k_\tau}(\tau),\tau]}_{k,k_\tau}\cdot p^{\tau-}_k\cdot\sum_{\substack{\nu\in(\alpha_{k_\tau}(\tau),\tau)\\ k_\nu=k}}\Delta t_\nu\\
\le\;& \frac{2e^{2\lambda'}}{p^{\tau-}_{k_\tau}}\sum_{k\ne k_\tau} L^{[\alpha_{k_\tau}(\tau),\tau]}_{k,k_\tau}\cdot p^{\tau-}_k. \qquad \text{(observe that the }\sum_\nu\Delta t_\nu\text{ term is at most 2)}
\end{aligned}$$

The summation $\sum_{k\ne k_\tau} L^{[\alpha_{k_\tau}(\tau),\tau]}_{k,k_\tau}\cdot p^{\tau-}_k$ above can be bounded as in (14), yielding an upper bound of $e^{2\lambda'}\cdot x_{k_\tau}(\mathbf{p}^{\tau-})$. ◀

To conclude, by (11) and Lemma 6, it suffices to have:

$$\frac{1}{\lambda p^{\tau-}_{k_\tau}}\cdot\max\{1,\tilde{z}_{k_\tau}\} \;=\; \Gamma^\tau_{k_\tau} \;\ge\; \max\left\{21c_1e^{4\lambda(\lambda+1)}\;,\;\frac{4}{c_1}\cdot e^{8\lambda(\lambda+1)}\right\}\cdot\frac{x_{k_\tau}(\mathbf{p}^{\tau-})}{p^{\tau-}_{k_\tau}},$$

or equivalently, $\lambda \cdot \max\left\{21c_1 e^{4\lambda(\lambda+1)}\ ,\ \frac{4}{c_1}\cdot e^{8\lambda(\lambda+1)}\right\} \leq \frac{\max\{1,\tilde{z}_{k_\tau}\}}{x_{k_\tau}(\mathbf{p}^{\tau-})}$. The minimum possible value of the RHS above is $1/(2e^{2\lambda(\lambda+1)})$. Thus, we need that

$$\lambda \cdot \max\left\{42c_1 e^{6\lambda(\lambda+1)}\ ,\ \frac{8}{c_1}\cdot e^{10\lambda(\lambda+1)}\right\} \leq 1.$$

We choose c_1 such that the two parameters in the max are equal, i.e., $c_1 = \frac{2}{\sqrt{21}}\cdot e^{2\lambda(\lambda+1)}$. Then the above inequality reduces to $4\sqrt{21}\cdot\lambda\cdot e^{8\lambda(\lambda+1)} \leq 1$; $\lambda \leq 1/25.5$ suffices.

References

1 Carlos Alós-Ferrer and Nick Netzer. The logit-response dynamics. *Games and Economic Behavior*, 68(2):413–427, 2010.

2 Kenneth J. Arrow, H. D. Block, and Leonid Hurwicz. On the stability of competitive equilibrium, ii. *Econometrica*, 27(1):82–109, 1959.

3 Petra Berenbrink, Tom Friedetzky, Leslie Ann Goldberg, Paul W. Goldberg, Zengjian Hu, and Russell A. Martin. Distributed selfish load balancing. *SIAM J. Comput.*, 37(4):1163–1181, 2007. doi:10.1137/060660345.

4 Benjamin Birnbaum, Nikhil R. Devanur, and Lin Xiao. Distributed algorithms via gradient descent for fisher markets. In *Proceedings of the 12th ACM Conference on Electronic Commerce*, EC '11, pages 127–136. ACM, 2011. doi:10.1145/1993574.1993594.

5 Lawrence E. Blume. The statistical mechanics of strategic interaction. *Games and Economic Behavior*, 5(3):387–424, 1993.

6 Bernard Chazelle. Natural algorithms. In *SODA*, pages 422–431, 2009.

7 Bernard Chazelle. The dynamics of influence systems. In *FOCS*, pages 311–320, 2012.

8 X. Chen, D. Dai, Y. Du, and S. H. Teng. Settling the complexity of arrow-debreu equilibria in markets with additively separable utilities. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 273–282, 2009. doi:10.1109/FOCS.2009.29.

9 Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *J. ACM*, 56(3):14:1–14:57, may 2009.

10 Xi Chen, Dimitris Paparas, and Mihalis Yannakakis. The complexity of non-monotone markets. *J. ACM*, 64(3):20:1–20:56, 2017. doi:10.1145/3064810.

11 Yun Kuen Cheung. *Analyzing Tatonnement Dynamics in Economics Markets*. PhD thesis, Courant Institute of Mathematical Sciences, New York University, Proquest Dissertations Publishing, 2014. Available at https://cs.nyu.edu/media/publications/cheung_yunkuen.pdf.

12 Yun Kuen Cheung and Richard Cole. Amortized analysis on asynchronous gradient descent. *CoRR*, abs/1412.0159, 2014.

13 Yun Kuen Cheung, Richard Cole, and Nikhil Devanur. Tatonnement beyond gross substitutes? Gradient descent to the rescue. In *STOC*, pages 191–200, 2013. Full version available at https://cims.nyu.edu/~ykcheung/publication/STOC13_full_paper.pdf. doi:10.1145/2488608.2488633.

14 Yun Kuen Cheung, Richard Cole, and Ashish Rastogi. Tatonnement in ongoing markets of complementary goods. In *EC*, pages 337–354, 2012. doi:10.1145/2229012.2229039.

15 Yun Kuen Cheung, Richard Cole, and Yixin Tao. Dynamics of distributed updating in fisher markets. In *Proceedings of the 2018 ACM Conference on Economics and Computation, Ithaca, NY, USA, June 18-22, 2018*, EC '18, pages 351–368, 2018. doi:10.1145/3219166.3219189.

16 Yun Kuen Cheung, Richard Cole, and Yixin Tao. A unified approach to analyzing asynchronous coordinate descent — standard and partitioned. *Submitted*, 2018.

17 Bruno Codenotti, Benton McCune, and Kasturi Varadarajan. Market equilibrium via the excess demand function. In *STOC*, pages 74–83, 2005. doi:10.1145/1060590.1060601.

18 Bruno Codenotti, Amin Saberi, Kasturi Varadarajan, and Yinyu Ye. Leontief economies encode nonzero sum two-player games. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 659–667. Society for Industrial and Applied Mathematics, 2006.

19 Richard Cole and Lisa Fleischer. Fast-converging tatonnement algorithms for one-time and ongoing market problems. In *STOC*, pages 315–324, 2008. doi:10.1145/1374376.1374422.

20 Richard Cole, Lisa Fleischer, and Ashish Rastogi. Discrete price updates yield fast convergence in ongoing markets with finite warehouses. *CoRR*, 2010.

21 Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a nash equilibrium. *SIAM J. Comput.*, 39(1):195–259, 2009.

22 Nikhil R. Devanur, Christos H. Papadimitriou, Amin Saberi, and Vijay V. Vazirani. Market equilibrium via a primal-dual algorithm for a convex program. *J. ACM*, 55(5):22:1–22:18, 2008.

23 Akitaka Dohtani. Global stability of the competitive economy involving complementary relations among commodities. *Journal of Mathematical Economics*, 22(1):73–83, 1993.

24 Ran Duan and Kurt Mehlhorn. A combinatorial polynomial algorithm for the linear arrow-debreu market. *Inf. Comput.*, 243:112–132, 2015. doi:10.1016/j.ic.2014.12.009.

25 Krishnamurthy Dvijotham, Yuval Rabani, and Leonard J. Schulman. Convergence of incentive-driven dynamics in fisher markets. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 554–567. Society for Industrial and Applied Mathematics, 2017.

26 Eyal Even-Dar, Alexander Kesselman, and Yishay Mansour. Convergence time to nash equilibrium in load balancing. *ACM Trans. Algorithms*, 3(3):32, 2007. doi:10.1145/1273340.1273348.

27 K. Jain. A polynomial time algorithm for computing the arrow-debreu market equilibrium for linear utilities. In *Forty Fifth Annual IEEE Symposium on Foundations of Computer Science*, FOCS'04, pages pp. 286–294, Rome, Italy, 2004.

28 Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. In *STOC*, pages 911–920, 2013.

29 Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *FOCS*, pages 147–156, 2013.

30 C. E. Lemke and J. T. Howson Jr. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, 1964.

31 Jure Leskovec, Lars Backstrom, and Jon M. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *KDD*, pages 497–506, 2009.

32 Jason R. Marden and Jeff S. Shamma. Revisiting log-linear learning: Asynchrony, completeness and payoff-based implementation. *Games and Economic Behavior*, 75(2):788–808, 2012.

33 James B. Orlin. Improved algorithms for computing Fisher's market clearing prices. In *Proceedings of the Forty Second Annual ACM Symposium on Theory of Computing*, STOC'10, pages 291–300, 2010.

34 Siddharth Pal and Richard J. La. Simple learning in weakly acyclic games and convergence to Nash equilibria, 2015. URL: http://www.ece.umd.edu/~hyongla/PAPERS/ALLERTON15.pdf.

35 Christos H. Papadimitriou and Mihalis Yannakakis. An impossibility theorem for price-adjustment mechanisms. *PNAS*, 5(107):1854–1859, 2010.

36 Hirofumi Uzawa. Walras' tatonnement in the theory of exchange. *Review of Economic Studies*, 27(3):182–194, 1960.
37 Vijay V. Vazirani and Mihalis Yannakakis. Market equilibrium under separable, piecewise-linear, concave utilities. *J. ACM*, 58(3):10:1–10:25, 2011.
38 Léon Walras. *Eléments d'Economie Politique Pure*. Corbaz, 1874. (Translated as: *Elements of Pure Economics*. Homewood, IL: Irwin, 1954.).
39 Fang Wu and Li Zhang. Proportional response dynamics leads to market equilibrium. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 354–363. ACM, 2007. doi:10.1145/1250790.1250844.
40 Yinyu Ye. A path to the arrow-debreu competitive market equilibrium. *Math. Program.*, 111(1-2):315–348, 2008. doi:10.1007/s10107-006-0065-5.
41 Li Zhang. Proportional response dynamics in the fisher market. *Theor. Comput. Sci.*, 412(24):2691–2698, 2011.

Cycles to the Rescue! Novel Constraints to Compute Maximum Planar Subgraphs Fast

Markus Chimani
Theoretical Computer Science, Osnabrück University, Germany
markus.chimani@uni-osnabrueck.de
https://orcid.org/0000-0002-4681-5550

Tilo Wiedera
Theoretical Computer Science, Osnabrück University, Germany
tilo.wiedera@uni-osnabrueck.de
https://orcid.org/0000-0002-5923-4114

Abstract

The NP-hard *Maximum Planar Subgraph* problem asks for a planar subgraph H of a given graph G such that H has maximum edge cardinality. For more than two decades, the only known non-trivial exact algorithm was based on integer linear programming and Kuratowski's famous planarity criterion. We build upon this approach and present new constraint classes – together with a lifting of the polyhedron – to obtain provably stronger LP-relaxations, and in turn faster algorithms in practice. The new constraints take Euler's polyhedron formula as a starting point and combine it with considering cycles in G. This paper discusses both the theoretical as well as the practical sides of this strengthening.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization, Mathematics of computing → Graph theory, Theory of computation → Linear programming

Keywords and phrases algorithm engineering, graph algorithms, integer linear programming, maximum planar subgraph

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.19

Related Version `https://arxiv.org/abs/1806.08283`

Funding Supported by the German Research Foundation (DFG) project CH 897/2-1.

1 Introduction

The NP-hard *Maximum Planar Subgraph* (MPS) problem is an established question in graph theory, already discussed in the classical textbook by Garey and Johnson [14, 20]. Given a graph G, we ask for a largest subset $F \subseteq E(G)$ of edges such that F induces a planar graph. By contrast, the closely related *maximal* planar subgraph problem asks for a set of edges that we cannot extend without violating planarity and is trivially solvable in polynomial time. The inverse measure of MPS that counts the minimum number of edges that must be removed to obtain a planar subgraph, is called the *skewness* of G and denoted by $\mathit{skew}(G)$.

There are several reasons why this problem has received a good deal of attention: Graph theoretically, skewness is a very natural and common measure of non-planarity (like crossing number or genus). Algorithmically, finding a large planar subgraph is central to the planarization method [1, 5] that is heavily used in graph drawing: one starts with a large (favorably maximum) planar subgraph and re-inserts the deleted edges, typically to obtain a low number of overall crossings. In fact, this gives an approximation of the crossing number

© Markus Chimani and Tilo Wiedera;
licensed under Creative Commons License CC-BY

with ratio roughly $\mathcal{O}(\Delta \cdot \mathit{skew}(G))$ [9], where Δ is the maximum node degree. Furthermore, several graph problems become easier when the input's skewness is small or constant. E.g., we can compute a maximum flow in time $\mathcal{O}(\mathit{skew}(G)^3 \cdot |V(G)| \log |V(G)|)$ [16][1] – the same runtime complexity as on planar graphs if the skewness is constant.

There are several practical heuristic approaches to tackle the problem [10]. However, MPS is MaxSNP-hard, i.e., there is an upper bound < 1 on the obtainable approximation ratio unless $P = NP$ [2], and there are further limits known for specific algorithmic approaches [3,7]. Already a spanning tree gives an approximation ratio of 1/3, the best known ratio is 4/9 [2], and only recently a practical 13/33-approximation algorithm emerged [3].

Considering exact algorithms, options are scarce. Over two decades ago, an integer linear program based on Kuratowski's characterization of planarity was introduced in [23], which remained the only non-trivial exact algorithm. Only very recently, [8] showed the existence of potentially feasible alternatives to the Kuratowski-based approach, but the former still constitutes the practically by far most efficient (and theoretically most thoroughly explored) model. All known ILP models (including those discussed in this paper) can also directly solve the *weighted* MPS, i.e., identify the heaviest planar subgraph w.r.t. given edge weights.

Contribution. In this paper, we strengthen the Kuratowski model by introducing new constraints and supplementary variables, based on analyzing the cycles occurring in the solutions; see Section 3. In particular, we show in Section 3.2 that starting with the original Kuratowski model and considering cycles of growing lengths yields a natural hierarchy of ever stronger LP-relaxations. In Section 3.3, we establish additional constraint classes using our cycle variables to further strengthen the LP-relaxations, both theoretically and practically. We show the latter property in an experimental evaluation in Section 4. We skip the proofs of some lemmata, in which case we mark the lemma with '$\star$'.

2 Preliminaries

Graph Notation. Our non-planar input graph is called G. Generally, we consider an undirected graph H, with nodes $V(H)$ and edges $E(H)$, which are cardinality-2 subsets of $V(H)$. We use $\delta_H(v)$ to denote all edges incident to node v in H and define the node degree $\deg_H(v) := |\delta_H(v)|$. If H is a subgraph of G, we write $H \subseteq G$. A (sub)graph is a *cycle* if it is connected and all its nodes have degree 2. The *girth* $\gamma(H)$ of H is the length of its smallest cycle. The union of two (non-disjoint) graphs H_1, H_2 is denoted by $H_1 \sqcup H_2 := (V(H_1) \cup V(H_2), E(H_1) \cup E(H_2))$. For $W \subseteq V(H)$ and $F \subseteq E(H)$ we define node- and edge-induced subgraphs $H[W] := (W, \{e \in E(H) \mid e \subseteq W\})$ and $H[F] := (\bigcup_{e \in F} e, F)$, respectively. We further use $H - e := H[E(H) \setminus \{e\}]$.

Given a planar drawing $\mathcal{D}$ of some planar graph H, the cyclic adjacency order around each node in $\mathcal{D}$ defines an *embedding* π of H. The disjoint regions bounded by edges in $\mathcal{D}$ correspond to the *faces* of π; the infinite region, bounded only on the inside, is called *outer face*. The *degree* $\deg(f)$ of any face f is the number of *half-edges* ("sides" of edges) that occur on the boundary of f; a bridge occurs twice on the same face.

Linear Programming. A *Linear Program* (LP) is a vector $c \in \mathbb{R}^d$ and a set of linear inequalities (*constraints*) that define a polyhedron P in $\mathbb{R}^d$; we ask for an element $x \in P$ that maximizes $c^\intercal x$. An *Integer Linear Program* (ILP) additionally requires the components

[1] [16] considers the crossing number; the algorithm trivially works also for the stronger parameter skewness.

of x to be integral. For a given problem, one can establish different ILPs, so-called *models*. To solve an ILP model, one uses branch-and-bound, where dual bounds are obtained from (fractional) solutions to the *LP-relaxation*, i.e., the ILP without the integrality requirements. Clearly, strong such LP-bounds are desired. We say a model N is *at least as strong* as a model M, if N's LP-relaxation gives no worse bounds than M's. We say N is *stronger* than M if, additionally, there is an instance where N gives a strictly better bound. If, in this case, N arises from M by adding some constraints C, we say C *strengthen* M.

It is often beneficial to consider only a relevant subset of constraints in the solving process, in particular when the class of constraints is (exponentially) large. The procedure is referred to as *separation*. We employ it on (fractional) LP-solutions for selected constraint classes.

Kuratowski Model (ε-Model). The following ILP is due to Mutzel [23]. Jünger and Mutzel showed that both constraint classes below form facets of the planar subgraph polytope [17]. We use solution variables $s_e \in \{0,1\}$ (for all $e \in E(G)$) that are 1 if and only if edge e is deleted, i.e., *not* in the planar subgraph. (In [23], equivalent variables $x_e := 1 - s_e$ are used.) The objective minimizes the skewness – thus maximizes the planar subgraph – and is given by

$$\min \sum\nolimits_{e \in E(G)} w(e)\, s_e.$$

Thereby, we may consider edge weights w; they are 1 in case of the traditional unweighted MPS problem. For a given subset $F \subseteq E(G)$ of edges, we define $s(F) := \sum_{e \in F} s_e$ as a shorthand. We can always use Euler's bound on the number of edges in planar graphs:

$$s\big(E(G)\big) \geq |E(G)| - (3n-6) + \mathbb{1}_{G \text{ is bipartite}}(n-2). \tag{1}$$

By Kuratowski's theorem [19], a graph is planar if and only if it neither contains a subdivision of a K_5 nor of a $K_{3,3}$. Hence, it suffices to ask for any member of the (exponentially large) set $\mathcal{K}(G)$ of all Kuratowski subdivisions that at least one of its edges is deleted:

$$s\big(E(K)\big) \geq 1 \qquad\qquad \forall K \in \mathcal{K}(G). \tag{2}$$

Clearly, (2) are too many constraints to use all explicitly. Instead, we identify a sufficient subset of constraints via a (heuristic) separation procedure: we round the fractional solution and obtain a graph that can be tested for planarity. If it is non-planar, we extract a Kuratowski subdivision. This method does neither guarantee to always find a violated constraint if there is any, nor that the identified subdivision in fact corresponds to a *violated* Kuratowski constraint. Still, since it has these guarantees on integral solutions, it suffices to obtain an exact algorithm. Over the years, the performance of this approach was improved by strong preprocessing [4], finding *multiple* Kuratowski subdivision in linear time [11], and strong primal heuristics [10]. We use all these identically in all considered algorithms.

The Kuratowski-model forms the basis of our extensions. As such, we denote it, without any of the below extensions, by 'ε-model'.

3 Stronger Constraints Based on Cycles

We now present new constraints for the planar subgraph polytope (or a lifted version thereof). All but the first class require the introduction of new variables based on cycles, leading to the *cycle model*. For each constraint class we first give some motivation and intuition for its feasibility, before discussing its technical details. We then describe – provided the class is large – separation routines that quickly identify violated constraints, and usually show that it strengthens our ILP model.

Generalized Euler Constraints. We know from [17] that inequality $|E(G)| \le 2|V(G)| - 4$ is facet-defining for complete biconnected graphs. We are interested in a class of similar constraints for dense subgraphs with large girth. The following lemma is folklore:

▶ **Lemma 1.** A planar graph G has at most $\big(|V(G)| - 2\big)\gamma(G)/\big(\gamma(G) - 2\big)$ edges.

Proof. Let $n := |V(G)|$, $m := |E(G)|$, and π denote an embedding of G. For any face of π we require at least $\gamma(G)$ half-edges. Thus, the number f of faces in π is bounded by $f \le 2m/\gamma(G)$. Using Euler's formula, we obtain $n - m + (2m/\gamma(G)) \le 2$, the claimed results follows when solving for m. ◀

We can thus derive a feasible *generalized Euler constraint* for any subgraph $H \subseteq G$:

$$|E(H)| - s\big(E(H)\big) \le \big(|V(H)| - 2\big)\gamma(H)/\big(\gamma(H) - 2\big) \qquad \forall H \subseteq G \tag{3}$$

We note that this bound can sometimes be improved: for constraints (3) to be satisfied with equality it is necessary that $V(H) \equiv 2 \pmod{\gamma(H) - 2}$ if $\gamma(H)$ is odd and $V(H) \equiv 2 \ \big(\mathrm{mod}\ (\gamma(H) - 2)/2\big)$ otherwise [13]. However, we did not implement this in our algorithms.

▶ **Lemma 2.**★ The generalized Euler constraints (3) strengthen the ε-model.

Proof sketch. $K_{3,3,1}$ contains a $K_{3,4}$ that prohibits the otherwise feasible solution 3/2. ◀

We separate constraints (3) heuristically by seeking dense, high-girth subgraphs using two different methods. First, using the current fractional solution, we assign weight $1 - s_e$ to each edge e and approximate a maximum cut [22, Section 6.3], obtaining a girth-4 subgraph. If (after postprocessing, see below) this does not yield a violated constraint, we try a second method: We set a target girth μ and iteratively add edges in ascending order of their LP-value to an initially empty graph, while updating the shortest paths between all node pairs. Upon adding an edge e, we check whether e would create a cycle of length $< \mu$, in which case we discard e instead. We may repeat this process for different values of μ. After each of the above attempts, we apply a postprocessing: Let H denote a girth-μ subgraph. The *contribution* of a node $v \in V(H)$ is defined by $|\delta_H(v)| - \sum_{e \in \delta_H(v)} s_e - \mu/(\mu - 2)$. We iteratively remove nodes with negative contribution from H. In particular, this will remove all degree-1 nodes.

3.1 Cycle Model

We now want to bound the number of edges in the planar subgraph by the number of its small faces. Even though compelling from a theoretical standpoint, it is infeasible to generate all potential faces of all planar subgraphs of a given graph (already for bounded length). However, we know that traversing the border of any face of a spanning subgraph H traverses at least one cycle if H is not a tree. We will relate the number of small faces in any planar subgraph of a graph G to the number of small cycles in G. One may also view this as a way to further generalize Euler constraints: many – in particular sparse – graphs have low girth only due to very few cycles of small length.

We may assume any (maximal) primal solution to be connected and non-outerplanar as it could be trivially improved otherwise. Also observe that we cannot require faces to uniquely map to cycles in general. Consider for example a cycle graph (two faces with the same cycle) or a non-biconnected graph (each cut-node occurs twice in at least one face; cycles contain nodes at most once) Note that there are biconnected graphs that have no biconnected MPS.

▶ **Lemma 3.** For every connected, planar but non-outerplanar subgraph H of G, there exists an embedding of H such that we can assign a unique cycle α to every face f where all edges of α occur on the boundary of f.

Proof. Let $H \subset G$ be as defined in the claim. There exists some biconnected component $B^\star$ of H that is neither a cycle nor an edge since H is not outerplanar. Choose an embedding of H and pick some face of $B^\star$ as the outer one. For every biconnected component B that is not just an edge, we iterate over the inner faces of B. Each inner face f of B directly corresponds to a cycle as a biconnected graph contains neither cut-nodes nor bridges. (Observe that an inner face of B might in fact be much larger in H since we ignore other components nested in this face.) Ultimately, we assign the cycle induced by the outer face in H to the (last remaining) outer face. Since $B^\star$ is not a cycle we do not assign any cycle twice. ◀

We denote the number of faces whose degree satisfies some property $\mathcal{P}$ by $f_\mathcal{P}$.

▶ **Lemma 4.** Given a connected, planar graph H on n nodes and m edges, for each embedding of H with exactly $f_{=d}$ faces of degree $d \in \{3, 4 \dots, 2m\}$, we have

$$m = 3n - 6 - \sum_{d=3}^{2m} (d-3) f_{=d}. \tag{4}$$

Proof. Every face in any embedding of H has degree at least 3 and at most $2m$. For every face f of degree d we can add $d-3$ edges that split f into $d-2$ triangles without violating planarity. After performing this operation for each face we obtain a planar triangulated graph, i.e., a graph that has exactly $3n-6$ edges. ◀

Let $\mathcal{C}_d(G)$ denote all cycles of length d in G. We set $D \geq 3$ to the *maximum cycle length* that we want to investigate; this parameter will control the number of additionally generated variables. Let $\mathcal{C}_{\leq D}(G)$ denote the set of cycles with length at most D. For every cycle $\alpha \in \mathcal{C}_{\leq D}(G)$ we generate a variable $c_\alpha \in \{0, 1\}$.[2] We force such a variable to 0 if any edge of the respective cycle is removed and allow at most two cycles per edge in the MPS:

$$\sum_{\alpha \in \mathcal{C}_{\leq D}(G)\colon\, e \in E(\alpha)} c_\alpha \leq 2\,(1 - s_e) \qquad\qquad \forall e \in E(G) \tag{5}$$

Note that constraints (5) resemble the requirement for each edge to appear in at most two faces (subject to Lemma 3). We discuss its correctness below. Let $c(d) := \sum_{\alpha \in \mathcal{C}_d(G)} c_\alpha$.

▶ **Lemma 5.** For every connected, planar but non-outerplanar subgraph H of G, there exists an embedding π of H and a feasible assignment w.r.t. (5) of cycle variables such that for each $d \leq D$ the number $f_{=d}$ of faces with degree d in π is bounded from above by

$$f_{\leq d} \leq \sum_{k=3}^{d} c(k), \quad \text{or equivalently} \quad f_{=d} \leq \sum_{k=3}^{d} c(k) - f_{<d}.$$

Proof. We assign cycle variables following the proof of Lemma 3. Hence, there is a unique cycle variable assigned to each face such that the length of the cycle is at most the degree of its face. The variable assignment is feasible since we pick only edges contained in H and pick at most two cycles incident with any such edge. ◀

▶ **Theorem 6.** *For any maximum planar subgraph of a graph G on n nodes and m edges there exists a feasible variable assignment that satisfies* (5) *and the* cycle constraint

$$(D-1)\big(m - s(E(G))\big) \leq (D+1)(n-2) + \sum_{d=3}^{D} (D+1-d)c(d). \tag{6}$$

[2] Intuitively, we want $c_\alpha = 1$ if and only if α is (part of) a face, see below for details. In terms of correctness, we need not but can actively force these variables to be binary, cf. Section 4.

Proof. Starting with (4) on any connected, planar subgraph of G that has $m - s(E(G))$ edges, we relax the equality by using the same coefficient for all faces of large degree as in

$$m - s(E(G)) \leq 3n - 6 - \sum\nolimits_{d=3}^{D}(d-3)f_{=d} - (D-2)f_{>D}.$$

By replacing $f_{>D}$ in Euler's formula, $(f_{>D} + f_{\leq D}) + n - \big(m - s(E(G))\big) = 2$, we obtain

$$(D-1)\big(m - s(E(G))\big) \leq (D+1)(n-2) + \sum\nolimits_{d=3}^{D}(D+1-d)f_{=d}. \tag{7}$$

The claimed cycle constraint is finally obtained by applying Lemma 5 to iteratively replace $f_{=D'}$ for $D' = D, D-1, \ldots, 4, 3$ by the upper bound (note that $f_{<3} = 0$), as sketched below for the (generalized, iteratively re-appearing) rightmost summand of (7):

$$\sum\nolimits_{d=3}^{D'}(D'+1-d)f_{=d} \leq \sum\nolimits_{d=3}^{D'-1}\big((D'-1)+1-d\big)f_{=d} + \sum\nolimits_{d=3}^{D'}c(d)$$ ◀

3.2 Relaxations and D-Hierarchy

We now turn our attention to LP-relaxations of the cycle model. We show that there is a hierarchy of gradually stronger LPs induced by the maximum cycle length D. Let the *cycle model* CM_D consist of the ε-model, the cycle variables for cycle lengths up to D, and the corresponding constraints (5),(6). If $D = 2$ were allowed, CM_2 would be exactly the ε-model.

▶ **Lemma 7.** For any solution to the relaxation of CM_D, it holds that

$$\sum\nolimits_{d=3}^{D}(d-2)c(d) \leq 2n-4.$$

Proof. Assume the contrary, $\sum_{d=3}^{D}(d-2)c(d) > 2n-4$. It follows that $\sum_{d=3}^{D} dc(d) > 2n-4+2\sum_{d=3}^{D} c(d)$ and hence $m - s(E(G)) > n-2+\sum_{d=3}^{D} c(d)$ by the sum of constraints (5). Plugging this bound on the number of edges into the cycle constraint (6), we obtain $\sum_{d=3}^{D}(d-2)c(d) < 2n-4$, a contradiction. ◀

It is not immediately clear, that decreasing the maximum cycle length maintains LP-feasibility, as some variables are removed and the cycle constraint is replaced. By employing Lemma 7, we can show the following fact.

▶ **Lemma 8.*** Model CM_{D+1} is at least as strong as CM_D.

▶ **Lemma 9.** Model CM_{D+1} is stronger than CM_D.

Proof. Consider the complete graph K_k on $k \geq 5$ nodes. Pick any number $\mu \geq D+1$. We subdivide every edge of K_k using $\xi := \lfloor \mu/3 \rfloor$ additional nodes. The resulting graph K_k^μ has girth at least μ, i.e., it has no cycles of length $\leq D$. We observe that $skew(K_n^\mu) = skew(K_k) = k(k-1)/2 - 3k + 6$, independent of μ. We show that increasing the maximum cycle length from D to $D+1$ cuts off all previously optimal LP solution.

Since K_k^μ has girth μ there can be at most $(|V(K_k^\mu)| - 2)\mu/(\mu-2)$ edges in any planar subgraph. As there are no cycle variables, the cycle constraint (6) approaches this value from above for increasing D. Any feasible solution that tightly satisfies the cycle constraint is an optimal one. The Kuratowski constraints (2) on the other hand are already satisfied by deleting each edge partially with $s_e = 1/(9\xi)\ \forall e \in E(K_n^\mu)$, since each subdivision requires at least 9ξ edges, still allowing LP-solutions with value $k(k-1)/18$. ◀

Overall, increasing the maximum cycle length strengthens our LP relaxations (leading to fewer LP-computations), but this comes at the cost of increasing the variable space (leading to slower LP-computations). It is imperative to find a good trade-off between these two.

3.3 Strengthening the Cycle Model

We now extend the cycle model further by introducing new constraint classes. Only the first such extension requires yet additional variables.

Pseudo-Tree Extension. Observe that degree-1 nodes in the solution deteriorate the cycle constraint's bound: given a face f that contains a degree-1 node, we can set the variable of a cycle with length at most $\deg(f) - 2$ to 1. We introduce new variables $t_v \in \{0,1\}$ for all $v \in V(G)$ and $t_{vw} \in \{0,1\}$ for all $v, w \in V(G)$ with $\{v,w\} \in E(G)$. They label nodes and directed edges (*arcs*) as *pseudo-trees*: any node with at most one unlabeled neighbor (in particular any degree-1 node) is to be labeled. This can be achieved by:

$$t_{vw} + t_{wv} \leq 1 - s_{\{v,w\}} \qquad \forall \{v,w\} \in E \qquad (8)$$

$$\sum\nolimits_{w \in N(v)} t_{vw} \geq t_v \qquad \forall v \in V(G) \qquad (9)$$

$$t_v + \deg_G(v) - \sum\nolimits_{w \in N(v)} t_{wv} - \sum\nolimits_{w \in N(v)} s_{vw} \geq 2 \qquad \forall v \in V(G) \qquad (10)$$

Constraints (8) allow at most one tree-arc for any edge and none for deleted edges. We force tree nodes to propagate along one outgoing arc by constraints (9). Finally, constraints (10) label degree-1-nodes and nodes where all (but one) neighbor is labeled. Now, we may subtract $\sum_{v \in V(G)} t_v$ nodes (and the same number of edges) from (6) to obtain a stronger bound:

▶ **Corollary 10.** The *extended cycle constraint*, given below, is feasible.

$$(D-1)\big(m - s(E(G))\big) \leq (D+1)(n-2) + \sum\nolimits_{d=3}^{D} (D+1-d)c(d) - 2\sum\nolimits_{v \in V(G)} t_v \quad (11)$$

Alternatively, we may use a less sophisticated approach that does not model propagation but labels only degree-1-nodes. In this case, it suffices to add variables $t_v \in \{0,1\}$, $\forall v \in V(G)$, and constraints (10), assuming $\sum_{w \in N(v)} t_{wv} = 0$.

▶ **Lemma 11.*** The pseudo-tree extension, i.e., constraints (8)–(11) together with the t-variables, strengthens CM_3. This already holds for the approach without propagation.

Proof sketch. We use the graph given in Fig. 1a: any MPS of it has a degree-1 node. ◀

All following constraint classes deal with excluding combinations of cycles and paths that either induce non-planarity, or result in cycle-variables not assignable to any face in the planar subgraph (Lemma 5). They are independent of but compatible with the pseudo-tree extension.

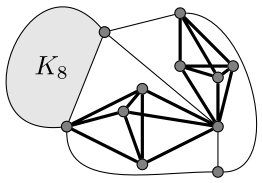
Cycle-Edge Constraints. Considering integral solutions and constraints (5), a cycle cannot be picked if any of its edges is deleted. W.r.t. fractional solutions we can additionally require

$$s_e + c_\alpha \leq 1 \qquad \forall \alpha \in \mathcal{C}_{\leq D}, e \in E(\alpha). \qquad (12)$$

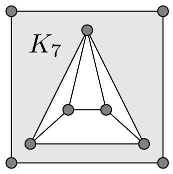
Although there are only $\mathcal{O}(D|\mathcal{C}_{\leq D}|)$ such constraints, preliminary benchmarks showed that adding all of them does not pay off. Instead, we straight-forwardly separate them by iterating over the edges of each cycle that has a non-zero variable.

▶ **Lemma 12.*** The cycle-edge constraints (12) strengthen CM_3.

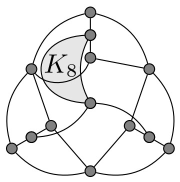
Proof sketch. Use a graph (Fig. 1b) that has 3 edges each incident to only 1 triangle. ◀



(a) Pseudo-Tree, Lemma 11



(b) Cycle-Edge, Lemma 12



(c) Two-Cycles-Path, Lemma 14

Figure 1 Graphs in strength proofs. Bold edges in **(a)** have large weight (or are edge bundles).

Two-Cycles-Path Constraints. Given two cycles α, β, we denote their set of inner nodes by $\nu(\alpha,\beta) := \{v \in V(\alpha) \cap V(\beta) \mid \delta_\alpha(v) = \delta_\beta(v)\}$. Let $\psi(\alpha,\beta)$ denote the set of non-empty paths that connect $\nu(\alpha,\beta)$ to $V(\alpha \sqcup \beta)$ without using any edge in $E(\alpha \sqcup \beta)$.

▶ **Lemma 13.** The two-cycles-path constraints, given below, are feasible.

$$s\big(E(p)\big) \geq c_\alpha + c_\beta - 1 \qquad \forall \alpha, \beta \in \mathcal{C}_{\leq D}; p \in \psi(\alpha,\beta) \tag{13}$$

Proof. Assume an embedding π of $\alpha \sqcup \beta$ where each of α, β corresponds to a face in π. By inserting p into π, we either split face α or face β. Hence, even in a supergraph of $\alpha \sqcup \beta \sqcup p$ two such faces cannot exist. Otherwise, if no such π exists, we have $1 \geq c_\alpha + c_\beta$. ◀

▶ **Lemma 14.**[⋆] The two-cycles-path constraints (13) strengthen CM_4.

Proof sketch. We use the graph of Fig. 1c as input. ◀

To identify violated two-cycles-path constraints, we consider each edge e. We collect the set $C(e) = \{\alpha \in \mathcal{C}_{\leq D} \mid e \in E(\alpha) \wedge c_\alpha > 0\}$, and check, for each pair $\alpha, \beta \in C(e)$, whether its sum of LP-values is > 1. If so, we compute the set of inner nodes $\nu := \nu(\alpha,\beta)$ and cache the result for future lookup. If $\nu \neq \varnothing$, we iteratively compute shortest paths following either of two patterns: the *combined* approach searches for shortest paths from ν to $V(\alpha \sqcup \beta) \setminus \nu$, whereas the *separate* one searches for paths from v to $V(\alpha \sqcup \beta) \setminus \{v\}$, separately for each $v \in \nu$. Note that the latter variant will always identify a violated constraint, if one exists, whereas the former ignores paths connecting two inner nodes. After identifying a new path p, an edge in $E(p)$ with maximal LP-value is discarded and the search at v starts anew.

We point out that there is a natural generalization of this constraint class by using k instead of only 2 cycles. If the k cycles fully enclose a common node v (like any 2 cycles enclose their inner nodes), any other path from v to the same block is forbidden.

Cycle-Two-Paths Constraints. We say that two paths p_1, p_2 are *conflicting w.r.t. a cycle* α if and only if they each start and end on nodes of $V(\alpha)$ but are otherwise disjoint from α and from one another, and p_2 connects the components of $\alpha[V(\alpha) \setminus V(p_1)]$.

▶ **Lemma 15.** The cycle-two-paths constraints, given below, are feasible.

$$s\big(E(p_1 \sqcup p_2)\big) \geq c_\alpha \qquad \forall \alpha \in \mathcal{C}_{\leq D}, \forall \text{ conflicting paths } p_1, p_2 \text{ w.r.t. } \alpha \tag{14}$$

Proof. Given an embedding π of α, we cannot insert both paths p_1, p_2 into the same face of π. Hence, we must split both faces in π. Consequently, no embedding of any supergraph of $\alpha \sqcup p_1 \sqcup p_2$ exists, where there is a face incident with all of α. ◀

While this constraint class may be stronger than the two-cycles-path constraints, we did not implement it: its separation is complex as we ask for two paths depending on each other.

Kuratowski-Cycle Constraints. Starting with a Kuratowski constraint, we can replace parts of its edges by cycles that contain them.

▶ **Lemma 16.** The Kuratowski-cycle constraints, given below, are feasible.

$$s(\{e \in E(K) \mid \forall \alpha \in C : e \notin E(\alpha)\}) \geq \sum\nolimits_{\alpha \in C} c_\alpha + 1 - |C| \qquad \forall K \in \mathcal{K}, C \subseteq \mathcal{C}_{\leq D} \qquad (15)$$

Proof. If $C = \varnothing$, we simply obtain a Kuratowski constraint. Assuming integrality and $C \neq \varnothing$, the right-hand side is 1 if all cycles in C are picked and ≤ 0 otherwise. In the former case, the edges of C, together with the remaining edges of K that are not contained in C contain a Kuratowski subdivision, and we need to remove an edge. ◀

▶ **Lemma 17.**★ The Kuratowski-cycle constraints (15) strengthen CM$_4$.

Proof sketch. We use the circulant on 16 nodes with jumps 1, 2, and 8 as input. ◀

For separation, we identify a Kuratowski subdivision K as for (2). We collect the set S of cycles with LP-value > 0 incident with K. For each cycle in S, we compute its *gain*, i.e., the increase in violation (or decrease in slack) when adding that cycle to C. While there are cycles with positive gain, we continue adding a cycle of S with maximal gain to C.

Cycle-Clique Constraints. Two cyclic orders $\pi, \bar{\pi}$ on a set X are *conflicting* if and only if $\pi \neq \bar{\pi}$ and $\pi \neq \text{reverse}(\bar{\pi})$. The restriction of π to $Y \subseteq X$ is denoted by π^Y. A cycle α induces a (up to reversal) unique cyclic order on its nodes $V(\alpha)$. Given two cycles α, β, let π_α, π_β be corresponding cyclic orders, and let $W := V(\alpha) \cap V(\beta)$ be the common nodes. We say that α and β are *conflicting* if and only if π_α^W and π_β^W are conflicting.

▶ **Lemma 18.** The cycle-clique constraints, given below, are feasible.

$$\sum\nolimits_{\alpha \in C} c_\alpha \leq 1 \qquad \forall C \subseteq \mathcal{C}_{\leq_D} \text{ s.t. all cycles in } C \text{ are pairwise conflicting} \qquad (16)$$

Proof. Consider any pair of conflicting cycles $\alpha, \beta \in C$ with π_α, π_β, and W defined as above. Since cyclic orders on three elements are unique up to reversal, we have $|W| \geq 4$. By transitivity there exists a set of exactly four common nodes $X \subseteq W$, such that π_α^X and π_β^X are conflicting. The graph on X where we add an edge vw if and only if v is adjacent to w in π_α^X or π_β^X is the K_4. Since the K_4 is not outerplanar, there can neither be a face in K_4 traversing all of X nor such a face in $\alpha \sqcup \beta$. ◀

We create the conflict graph H_C that contains a node for every cycle with LP-value > 0, cache the conflict information for each pair of cycles, and add constraints for maximal cliques in H_C. In a less sophisticated variant, we only add constraints for cliques of size two.

4 Experiments

All algorithms are implemented in `C++`, compiled with GCC 6.3.0, and use the OGDF (snapshot 2017-07-23) [6]. We use SCIP 4.0.1 for solving ILPs with CPLEX 12.7.1 as the underlying LP solver [21]. Each MPS-computation uses a single physical core of a Xeon Gold 6134 CPU (3.2 GHz) with a memory speed of 2666 MHz. We employ a time limit of 20 minutes and a memory limit of 8 GB per computation. Our instances and results, giving runtime and skewness (if solved), are available for download at `http://tcs.uos.de/research/mps`.

Instances and Algorithms. Analogously to the study [8], we consider three established real-world benchmark sets: *Rome* [12], *North* [24], and (a subset of) *SteinLib* [18]. We know from [7, 8] that random regular graphs (which are *expander graphs* with high probability) are especially hard to solve exactly. We use the same such instances as [8], but only consider graphs with ≤ 100 nodes, as no known exact algorithm solves larger instances. There are 20 graphs for each parameterization $(|V(G)|, \Delta) \in \{10, 20, 30, 50, 100\} \times \{4, 6, 10, 20, 40\}$, where $\Delta < |V(G)|$ is the node-degree. For tuning of ε (e.g., heap size in separation) we rely on the values identified in [15]. We use the notation below to encode algorithmic choices.

ε	Do not use any extensions but the basic Kuratowski algorithm [23].
e	Separate generalized Euler constraints (3).
c$\{r\}$	Add cycle constraints (5), (6), and variables with the minimal value for D such that there are variables for at least $100r$ cycles.
t$\{0\|1\}$	Use the pseudo-tree extension (8)–(11) with (=t1) or without (=t0) propagation.
i	Enforce integrality of variables for cycles and pseudo-trees.
s	Separate cycle-edge constraints (12).
w$\{0\|1\}$	Separate two-cycles-path constraints (13) using *combined* (=w0) or *separate* (=w1) approach. Also enables separation on cycle-clique constraints (16) for 2-cliques.
k	Separate Kuratowski-cycle constraints (15).
q	Separate cycle-clique constraints (16).

Note that instead of providing D explicitly, we specify a minimum number r' of cycle variables to be generated. We increment D while there are less than r' cycle variables.

Results. Table 1 shows the success rates (percentage of instances solved to proven optimality) and average runtime per instance of our algorithmic variants. For non-solved instances we assume the maximum runtime of 20 minutes – average runtimes are thus comparable only for algorithms that achieve roughly equal success rates. We group the variants by the number of used extensions and highlight variants that dominate their group in bold. The latter informs our choice of which variants to consider in the next group.

The separation of generalized Euler constraints is clearly beneficial only on the North graphs, but even there its improvements are marginal when compared to the cycle-based approach. The latter works very well in practice, for all instance sets. In particular (cf. Fig. 2), on *Rome* it allows us for the first time to compute the skewness of *all* instances. Using variant *c10 t0 i w0*, we are able to solve all but `grafo10958.98.lgr` within the 20 minute time frame; this last instance required 103 minutes. *North* still contains instances too hard to solve exactly (even when increasing runtime to a few days and memory to 32GB). Nonetheless, we now solve 3/4 of the previously unsolved *North* graphs within our strict limits. The second group of variants in Table 1 demonstrates that all of our extensions of the cycle model, in particular the pseudo-tree approach, improve upon success rate and runtime on all instance sets when applied to *c10*. As shown in the lower sections of the table, this does not always apply when comparing models that simultaneously use multiple extensions.

Table 2 details the relative improvement for each of the three most promising algorithm configurations over the state-of-the-art ε-model. We provide the success rate for the instances not solved by ε and give the average relative speed-up (i.e., the runtime of ε divided by that of variant X) over the instances solved by both ε and X. This common set is exactly those solved by ε, except for a single ε-solved *North*-instance not solved by *c10*. On *Rome*, the pure cycle model *c10* without any further extensions achieves the best speed-up; for the seemingly harder other instance sets, more sophisticated variants are worthwhile. Fig. 2 underlines that the success rate of the algorithms is strongly correlated to the instance's skewness.

Table 1 Overview of performance for algorithmic variants: success rate and avg. runtime.

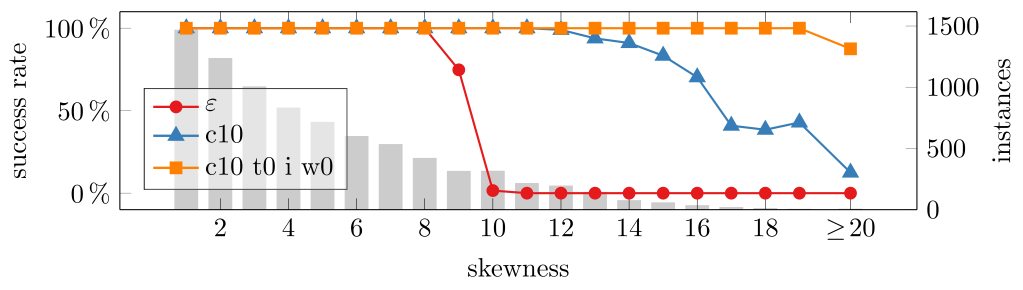
variant	Rome		North		Expanders		SteinLib	
	succ. [%]	time [s]	succ. [%]	time [s]	succ. [%]	time [s]	succ. [%]	time [s]
ε	85.71	198.42	73.76	325.31	34.74	800.38	9.52	1 085.94
e	85.56	199.41	77.78	273.29	35.00	803.91	9.52	1 085.93
c5	98.91	21.60	84.40	201.42	53.95	567.52	31.43	859.40
c10	**99.14**	**18.10**	**84.63**	**195.35**	54.47	562.81	**32.38**	**853.57**
c20	**99.14**	19.58	83.92	197.86	**55.00**	573.82	31.43	861.47
c10 i	99.89	5.52	88.89	156.99	56.58	538.81	31.43	841.88
c10 s	99.79	6.66	88.42	165.54	**58.68**	**515.13**	35.24	821.00
c10 t0	**99.95**	**3.36**	92.43	112.82	57.37	535.46	**37.14**	789.26
c10 t1	99.92	3.74	**93.14**	**111.94**	56.32	539.04	**37.14**	**785.89**
c10 w0	99.79	7.07	87.23	165.36	55.53	549.94	31.43	837.52
c10 w1	99.82	6.63	86.52	179.48	55.00	553.38	31.43	833.81
c10 k	99.77	7.26	86.52	178.34	55.26	552.83	33.33	828.81
c10 q	99.73	7.51	85.82	185.84	55.53	550.55	31.43	841.77
c10 t0 i	99.95	3.22	93.14	109.61	57.37	529.93	38.10	782.72
c10 t0 s	**99.98**	3.08	**93.62**	**95.46**	**58.95**	**509.01**	**39.05**	**760.70**
c10 t0 w0	**99.98**	**2.75**	92.43	112.77	57.37	537.61	36.19	808.58
c10 t0 w1	**99.98**	2.92	92.20	109.37	57.11	537.76	37.14	780.83
c10 t0 k	99.92	3.57	92.67	104.55	56.84	535.97	38.10	785.46
c10 t0 q	99.95	3.58	92.67	109.71	57.37	538.19	37.14	789.05
c10 t1 i	99.96	3.32	92.91	106.39	57.11	533.51	37.14	788.52
c10 t1 s	**99.98**	**2.75**	92.43	112.77	58.68	537.61	37.14	808.58
c10 t1 w0	**99.98**	3.04	92.20	114.28	56.84	537.97	38.10	786.93
c10 t1 w1	**99.98**	3.19	91.96	113.03	56.84	540.39	37.14	783.09
c10 t1 k	99.92	3.65	92.20	112.61	56.84	538.91	37.14	784.30
c10 t1 q	99.92	3.86	93.14	113.89	56.05	540.23	37.14	788.07
c10 t0 i s	99.94	3.27	92.91	103.17	**58.95**	506.63	**40.00**	761.47
c10 t0 s w0	**99.98**	2.43	**93.85**	**91.66**	58.68	508.28	39.05	763.08
c10 t0 s w1	**99.98**	**2.29**	92.91	101.17	58.68	507.99	39.05	756.31
c10 t0 s k	99.96	3.03	93.38	98.06	58.68	504.54	39.05	765.08
c10 t0 s q	99.93	3.22	93.62	95.59	58.42	510.38	38.10	763.64
c10 t0 i w0	**99.99**	2.89	92.67	105.16	57.11	529.95	38.10	798.26
c10 t0 i s w0	99.96	2.72	**94.33**	**93.99**	**59.47**	**502.30**	**39.05**	**754.46**

Table 2 Relative improvement over ε for selected algorithmic variants. We give the the success rate over the instances unsolved by ε, and the avg. runtime ratio over the commonly solved instances.

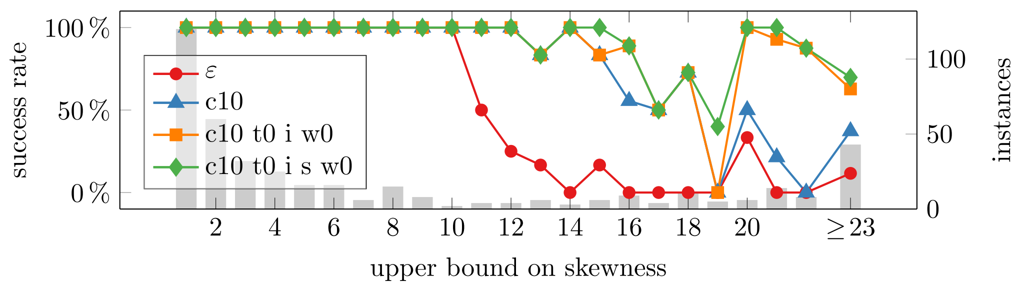
variant	Rome		North		Expanders		SteinLib	
	new [%]	speed-up	new [%]	speed-up	new [%]	speed-up	new [%]	speed-up
c10	93.98	**66.80**	42.34	21.45	30.24	13.96	25.26	**11.79**
c10 t0 i w0	**99.92**	60.85	72.07	28.59	34.27	12.68	31.58	6.79
c10 t0 i s w0	99.75	59.58	**78.38**	**34.03**	**37.90**	**23.21**	**32.63**	5.42

Table 3 Average number of cycle variables and average values for maximum cycle length D.

variant		Rome		North		Expanders		SteinLib	
	min # var	D	# var	D	# var	D	# var	D	# var
c5	500	9.51	627	7.34	689	5.43	2 075	5.80	881
c10	1 000	10.51	1 168	8.01	1 213	5.73	2 816	6.64	3 658
c20	2 000	11.51	2 175	8.55	2 048	6.47	7 774	7.09	4 785



(a) Solved Rome graphs by skewness. *c10 t0 i w0* solves all but 1 skew-22-graph within 20min.



(b) Solved North graphs by best upper bound on skewness.

Figure 2 Detailed success rates for selected algorithmic variants.

Table 3 lists the average number of generated cycle variables and the respective average values for D. We mention that instances with high D values typically generate *few* cycle variables, close to the lower bound. However, there is a large deviation in the number of generated cycle variables in any fixed instance set: some graphs contain less than the requested number of cycles whereas others already contain roughly 10 000 triangles.

5 Conclusion and Open Questions

For over two decades, the strongest ILP model for MPS has not been improved. In this paper we presented novel variables and constraints, based on cycles, to extend this model to finally obtain both a theoretically stronger model, as well as a more efficient algorithm in practice. We proved that there is a hierarchy of ever stronger LP-relaxations, induced by the maximal considered cycle length, and a rich set of further strengthening cycle-based constraint classes. For the first time, we are able to compute the skewness of *all* Rome graphs, solve 94% of the North graphs (compared to 74% by the ε-model), and solve 40% instead of only 10% of our SteinLib instances. Our extensions also help for the notoriously hard expander graphs.

Several of our proofs show the new constraint class's strength w.r.t. a low-D cycle model. We conjecture that most classes remain strengthening for high D, but to prove this, one has to find and argue infinite families of graphs with the LP-properties of our currently hand-crafted proof graphs. Furthermore, it is natural to ask if and which of the new constraint classes form facets in the (lifted) planar subgraph polytope.

A problem inherent to our approach arises on inputs of non-homogeneous density: G may have too dense subgraphs to raise D sufficiently, even when every planar subgraph of G contains large regions consisting of high-degree faces. Is there a practical way to generalize the cycle-based approach using an independent maximum cycle length *for each edge*?

References

1 Carlo Batini, Maurizio Talamo, and Roberto Tamassia. Computer Aided Layout of Entity Relationship Diagrams. *J. Syst. Soft.*, 4(2-3):163–173, 1984. doi:10.1016/0164-1212(84)90006-2.

2 Gruia Călinescu, Cristina Gomes Fernandes, Ulrich Finkler, and Howard Karloff. A Better Approximation Algorithm for Finding Planar Subgraphs. *J. Alg. in Cognition, Informatics and Logic*, 27(2):269–302, 1998. doi:10.1006/jagm.1997.0920.

3 Parinya Chalermsook and Andreas Schmid. Finding Triangles for Maximum Planar Subgraphs. In Sheung-Hung Poon, Md. Saidur Rahman, and Hsu-Chun Yen, editors, *WALCOM: Algorithms and Computation, 11th International Conference and Workshops, WALCOM 2017, Hsinchu, Taiwan, March 29-31, 2017, Proceedings.*, volume 10167 of *Lecture Notes in Computer Science*, pages 373–384. Springer, 2017. doi:10.1007/978-3-319-53925-6_29.

4 Markus Chimani and Carsten Gutwenger. Non-planar core reduction of graphs. *Discrete Mathematics*, 309(7):1838–1855, 2009. doi:10.1016/j.disc.2007.12.078.

5 Markus Chimani and Carsten Gutwenger. Advances in the Planarization Method: Effective Multiple Edge Insertions. *J. Graph Algorithms Appl.*, 13(3):729–757, 2012. doi:10.7155/jgaa.00264.

6 Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Karsten Klein, and Petra Mutzel. The Open Graph Drawing Framework (OGDF). In Roberto Tamassia, editor, *Handbook on Graph Drawing and Visualization*, pages 543–569. Chapman and Hall/CRC, 2013. URL: https://crcpress.com/Handbook-of-Graph-Drawing-and-Visualization/Tamassia/9781584884125.

7 Markus Chimani, Ivo Hedtke, and Tilo Wiedera. Limits of Greedy Approximation Algorithms for the Maximum Planar Subgraph Problem. In Veli Mäkinen, Simon J. Puglisi, and Leena Salmela, editors, *Combinatorial Algorithms - 27th International Workshop, IWOCA 2016, Helsinki, Finland, August 17-19, 2016, Proceedings*, volume 9843 of *Lecture Notes in Computer Science*, pages 334–346. Springer, 2016. doi:10.1007/978-3-319-44543-4_26.

8 Markus Chimani, Ivo Hedtke, and Tilo Wiedera. Exact Algorithms for the Maximum Planar Subgraph Problem: New Models and Experiments. In *17th International Symposium on Experimental Algorithms, SEA 2018, June 27-29, 2018, L'Aquila, Italy*, volume 103. LIPIcs, 2018. doi:10.4230/LIPIcs.SEA.2018.22.

9 Markus Chimani and Petr Hliněný. A tighter insertion-based approximation of the crossing number. *J. Comb. Optim.*, 33(4):1183–1225, 2017. doi:10.1007/s10878-016-0030-z.

10 Markus Chimani, Karsten Klein, and Tilo Wiedera. A Note on the Practicality of Maximal Planar Subgraph Algorithms. In Yifan Hu and Martin Nöllenburg, editors, *Proceedings of the 24th International Symposium on Graph Drawing and Network Visualization (GD 2016)*, volume abs/1609.02443. CoRR, 2016. doi:10.1007/978-3-319-50106-2_28.

11 Markus Chimani, Petra Mutzel, and Jens M. Schmidt. Efficient Extraction of Multiple Kuratowski Subdivisions. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *Graph Drawing, 15th International Symposium, GD 2007, Sydney, Australia, September 24-26, 2007. Revised Papers*, volume 4875 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2007. doi:10.1007/978-3-540-77537-9_17.

12 Giuseppe Di Battista, Ashim Garg, Giuseppe Liotta, Roberto Tamassia, Emanuele Tassinari, and Francesco Vargiu. An experimental comparison of four graph drawing algorithms. *Computational Geometry. Theory and Applications*, 7(5-6):303–325, 1997. 11th ACM Symposium on Computational Geometry (Vancouver, BC, 1995). doi:10.1016/S0925-7721(96)00005-3.

13 Manuel Fernández, Nicholas Sieger, and Michael Tait. Maximal Planar Subgraphs of Fixed Girth in Ramdom Graphs. *CoRR*, 2018. arXiv:1706.06202.

14 Michael R. Garey and David S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness.* W. H. Freeman and Co., San Francisco, Calif., 1979.

15 Ivo Hedtke. *Minimum Genus and Maximum Planar Subgraph: Exact Algorithms and General Limits of Approximation Algorithms.* PhD thesis, Osnabrück University, 2017. URL: `https://repositorium.ub.uos.de/handle/urn:nbn:de:gbv:700-2017082416212`.

16 Jan M. Hochstein and Karsten Weihe. Maximum s-t-flow with k crossings in $O(k^3 n \log n)$ time. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, SODA '07, pages 843–847, 2007.

17 Michael Jünger and Petra Mutzel. Maximum Planar Subgraphs and Nice Embeddings: Practical Layout Tools. *Algorithmica*, 16(1):33–59, 1996. `doi:10.1007/s004539900036`.

18 Thorsten Koch, Alexander Martin, and Stefan Voß. SteinLib: An updated library on steiner tree problems in graphs. Technical Report ZIB-Report 00-37, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, Berlin, 2000. URL: `http://elib.zib.de/steinlib`.

19 Kazimierz Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.

20 P. C. Liu and R. C. Geldmacher. On the deletion of nonplanar edges of a graph. In *Proceedings of the Tenth Southeastern Conference on Combinatorics, Graph Theory and Computing (Florida Atlantic Univ., Boca Raton, Fla., 1979)*, Congress. Numer., XXIII–XXIV, pages 727–738. Utilitas Math., Winnipeg, Man., 1979.

21 Stephen J. Maher, Tobias Fischer, Tristan Gally, Gerald Gamrath, Ambros Gleixner, Robert Lion Gottwald, Gregor Hendel, Thorsten Koch, Marco E. Lübbecke, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Sebastian Schenker, Robert Schwarz, Felipe Serrano, Yuji Shinano, Dieter Weninger, Jonas T. Witt, and Jakob Witzig. The SCIP Optimization Suite 4.0. Technical Report 17-12, ZIB, Takustr. 7, 14195 Berlin, 2017.

22 Michael Mitzenmacher and Eli Upfal. *Probability and computing - randomized algorithms and probabilistic analysis.* Cambridge University Press, 2005.

23 Petra Mutzel. *The maximum planar subgraph problem.* PhD thesis, Köln University, 1994.

24 Stephen C. North. 5114 directed graphs, 1995. Manuscript.

Parameterized Approximation Algorithms for Bidirected Steiner Network Problems

Rajesh Chitnis[1]
University of Warwick, UK
rajeshchitnis@gmail.com

Andreas Emil Feldmann[2]
Charles University, Prague, Czechia
feldmann.a.e@gmail.com

Pasin Manurangsi[3]
University of California, Berkeley, USA
pasin@berkeley.edu

Abstract

The Directed Steiner Network (DSN) problem takes as input a directed edge-weighted graph $G = (V, E)$ and a set $\mathcal{D} \subseteq V \times V$ of k demand pairs. The aim is to compute the cheapest network $N \subseteq G$ for which there is an $s \to t$ path for each $(s, t) \in \mathcal{D}$. It is known that this problem is notoriously hard as there is no $k^{1/4-o(1)}$-approximation algorithm under Gap-ETH, even when parameterizing the runtime by k [Dinur & Manurangsi, ITCS 2018]. In light of this, we systematically study several special cases of DSN and determine their parameterized approximability for the parameter k.

For the bi-DSN$_{\text{Planar}}$ problem, the aim is to compute a planar optimum solution $N \subseteq G$ in a bidirected graph G, i.e. for every edge uv of G the reverse edge vu exists and has the same weight. This problem is a generalization of several well-studied special cases. Our main result is that this problem admits a parameterized approximation scheme (PAS) for k. We also prove that our result is tight in the sense that (a) the runtime of our PAS cannot be significantly improved, and (b) it is unlikely that a PAS exists for any generalization of bi-DSN$_{\text{Planar}}$, unless FPT=W[1]. Additionally we study several generalizations of bi-DSN$_{\text{Planar}}$ and obtain upper and lower bounds on obtainable runtimes parameterized by k.

One important special case of DSN is the Strongly Connected Steiner Subgraph (SCSS) problem, for which the solution network $N \subseteq G$ needs to strongly connect a given set of k terminals. It has been observed before that for SCSS a parameterized 2-approximation exists when parameterized by k [Chitnis et al., IPEC 2013]. We show a tight inapproximability result: under Gap-ETH there is no $(2 - \varepsilon)$-approximation algorithm parameterized by k (for any $\epsilon > 0$). To the best of our knowledge, this is the first example of a W[1]-hard problem admitting a non-trivial parameterized approximation factor which is also known to be tight! Additionally we show that when restricting the input of SCSS to bidirected graphs, the problem remains NP-hard but becomes FPT for k.

2012 ACM Subject Classification Theory of computation → Routing and network design problems, Theory of computation → Fixed parameter tractability

Keywords and phrases Directed Steiner Network, Strongly Connected Steiner Subgraph, Parameterized Approximations, Bidirected Graphs, Planar Graphs

[1] Supported by ERC grant 2014-CoG 647557. Part of this work was done while at Weizmann Institute of Science, Israel and supported by Israel Science Foundation grant #897/13
[2] Supported by the Czech Science Foundation GAČR (grant #17-10090Y), and by the Center for Foundations of Modern Computer Science (Charles Univ. project UNCE/SCI/004).
[3] This work was done while the author was visiting Weizmann Institute of Science.

© Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi;
licensed under Creative Commons License CC-BY

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.20

Related Version A full version of this paper can be found in [12], `https://arxiv.org/abs/1707.06499`.

Acknowledgements We would like to thank Petr Kolman and Hans Raj Tiwary for giving valuable feedback on a draft of this paper.

1 Introduction

In this paper we study the Directed Steiner Network (DSN) problem,[4] in which a directed edge-weighted graph $G = (V, E)$ is given together with a set of k *demands* $\mathcal{D} = \{(s_i, t_i)\}_{i=1}^{k} \subseteq V \times V$. The aim is to compute a minimum cost (in terms of edge weights) network $N \subseteq G$ containing a directed $s_i \to t_i$ path for each $i \in \{1, \dots, k\}$. This well-studied problem has applications in network design [38], and for instance models the setting where nodes in a radio or ad-hoc wireless network connect to each other unidirectionally [10, 57].

The DSN problem is notoriously hard. First of all, it is NP-hard, and one popular way to handle NP-hard problems is to efficiently compute an *α-approximation*, i.e., a solution that is guaranteed to be at most a factor α worse than the optimum. For this paradigm we typically demand that the algorithm computing such a solution runs in polynomial time in the input size $n = |V|$. However for DSN it is known that even computing an $O(2^{\log^{1-\varepsilon} n})$-approximation is not possible [18] in polynomial time, unless NP $\subseteq$ DTIME($n^{\text{polylog}(n)}$). It is possible to obtain approximation factors $O(n^{2/3+\varepsilon})$ and $O(k^{1/2+\varepsilon})$ though [3, 9, 25]. For settings where the number k of demands is fairly small, one may aim for algorithms that only have a mild exponential runtime blow-up in k, i.e., a runtime of the form $f(k) \cdot n^{O(1)}$, where $f(k)$ is some function independent of n. If an algorithm computing the optimum solution with such a runtime exists for a computable function $f(k)$, then the problem is called *fixed-parameter tractable (FPT)* for parameter k. However it is unlikely that DSN is FPT for this well-studied parameter, as it is known to be W[1]-hard [31] for k. In fact one can show [14, 22] that under the *Exponential Time Hypothesis (ETH)* there is no algorithm computing the optimum in time $f(k) \cdot n^{o(k)}$ for any function $f(k)$ independent of n. ETH assumes that there is no $2^{o(n)}$ time algorithm to solve 3SAT [33, 34]. The best we can hope for is therefore a so-called *XP-algorithm* computing the optimum in time $n^{O(k)}$, and this was also shown to exist by Feldman and Ruhl [24].

None of the above algorithms for DSN seem satisfying though, either due to slow runtimes or large approximation factors, and this is hardly surprising given the problem's inherent complexity. To circumvent the hardness of the problem, one may aim for *parameterized approximations*, which have recently received increased attention for various problems (see e.g. [5, 8, 11, 13, 23, 26, 42, 44, 46, 49, 59, 62, 21, 4, 37]). In this paradigm an α-approximation is computed in time $f(k) \cdot n^{O(1)}$ for parameter k, where $f(k)$ again is a computable function independent of n. Unfortunately, a recent result by Dinur and Manurangsi [17][5] excludes significant improvements over the known polynomial time approximation algorithms [3, 9, 25], even if allowing a runtime parameterized in k. More specifically, no $k^{1/4-o(1)}$-approximation

[4] Also sometimes called Directed Steiner Forest. Note however that in contrast to the undirected Steiner Forest problem, an optimum solution to DSN is not necessarily a forest.

[5] In a previous version of this work, we showed that no $k^{o(1)}$-approximation is possible for DSN in time $f(k) \cdot n^{O(1)}$. This result in now subsumed by [17].

is possible in time $f(k) \cdot n^{O(1)}$ for any function $f(k)$ under the *Gap Exponential Time Hypothesis (Gap-ETH)*[6], which postulates that there exists a constant $\varepsilon > 0$ such that no (possibly randomized) algorithm running in $2^{o(n)}$ time can distinguish whether it is possible to satisfy all or at most a $(1-\varepsilon)$-fraction of clauses of any given 3SAT formula [16, 48].

Given these hardness results, the main question we explore is: what approximation factors and runtimes are possible for special cases of DSN when parametrizing by k? There are two types of standard special cases that are considered in the literature:

- Restricting the input graph G to some special graph class. A typical assumption for instance is that G is planar.[7]
- Restricting the pattern of the demands in $\mathcal{D}$. For example, one standard restriction is to have a set $R \subseteq V$ of *terminals*, a fixed *root* $r \in R$, and demand set $\mathcal{D} = \{(r,t) \mid t \in R\}$, which is the well-known Directed Steiner Tree (DST) problem.

In fact, an optimum solution to the DST problem is an arborescence (hence the name), i.e., it is planar. Thus if an algorithm is able to compute (an approximation to) the cheapest planar DSN solution in an otherwise unrestricted graph, it can be used for both the above types of restrictions: it can of course be used if the input graph is planar as well, and it can also be used if the demand pattern implies that the optimum must be planar. Taking the structure of the optimum solution into account has been a fruitful approach leading to several results on related problems, both for approximation and fixed-parameter tractability, from which we also draw some of the inspiration for our results (cf. Section 1.2). A main focus of our work is to systematically explore the influence of the structure of optimum solutions on the complexity of the DSN problem. Formally, fixing a class $\mathcal{K}$ of graphs, we define the $\mathrm{DSN}_{\mathcal{K}}$ problem, which asks for an optimum solution network $N \subseteq G$ for k given demands such that $N \in \mathcal{K}$. The $\mathrm{DSN}_{\mathcal{K}}$ problem has been implicitly studied in several results before for various classes $\mathcal{K}$, in particular when $\mathcal{K}$ contains either planar graphs, or graphs of bounded treewidth[8] (cf. Table 1).

Another special case we consider is when the input graph G is *bidirected*, i.e., for every edge uv of G the reverse edge vu exists in G as well and has the same weight as uv. This naturally captures the problem variant between the notoriously hard DSN problem on directed graphs and its undirected counterpart the Steiner Forest (SF) problem. As the former does not allow any $k^{1/4-o(1)}$-approximation in time $f(k) \cdot n^{O(1)}$ under Gap-ETH [17], while the latter is FPT [53, 27, 19] for parameter k, it is interesting to ask what happens between these two extremes. Bidirected graphs also model the realistic setting [10, 57, 61, 43] when the cost of transmitting from a node u to a node v in a wireless network is the same in both directions, which for instance happens if the nodes all have the same transmitter model.

We meticulously study several special cases of DSN resulting from the above restrictions, and prove matching upper and lower bounds on runtimes parameterized by k. We now give a brief overview of the studied problems emphasizing the main insights, and refer to Section 1.1 for a detailed exposition of our obtained results.

bi-$\mathrm{DSN}_{\mathrm{Planar}}$, i.e., the $\mathrm{DSN}_{\mathcal{K}}$ problem on bidirected inputs, where $\mathcal{K}$ is the class of planar graphs: For this problem we present our main result, which is that bi-$\mathrm{DSN}_{\mathrm{Planar}}$ admits a *parameterized approximation scheme (PAS)*, i.e., an algorithm that for any $\varepsilon > 0$ computes

[6] Gap-ETH follows from ETH given other standard conjectures, such as the existence of linear sized PCPs or exponentially-hard locally-computable one-way functions. See [8, 2] for more details.
[7] A directed graph is planar if the underlying undirected graph is.
[8] Here the *undirected* treewidth is meant, i.e., the treewidth of the underlying undirected graph.

a $(1+\varepsilon)$-approximation in $f(\varepsilon,k)\cdot n^{g(\varepsilon)}$ time for some functions f and g. We also prove that, unless FPT=W[1], no *efficient parameterized approximation scheme (EPAS)* exists, i.e., there is no algorithm computing a $(1+\varepsilon)$-approximation in $f(\varepsilon,k)\cdot n^{O(1)}$ time for any function f. Thus the runtime of our algorithm cannot be significantly improved.

BI-DSN, i.e., the DSN problem on bidirected inputs: The above PAS for the restricted BI-DSN$_{\text{PLANAR}}$ problem begs the question of whether a PAS also exists for any more general problems, such as BI-DSN. However we prove that BI-DSN does not admit a PAS under Gap-ETH. At the same time it is not too hard to obtain constant approximations in parameterized or polynomial time, given known algorithms for SF. When aiming for optimum solutions however, surprisingly we can show that BI-DSN is almost as hard as DSN (with almost-matching runtime lower bound under ETH). Thus the complexity of the in-between bidirected setting resembles that of the directed setting in terms of FPT algorithms, while in terms of approximations it is more similar to the undirected setting.

Apart from the DST problem, another well-studied special case of DSN with restricted demands is when the demand pairs form a cycle, i.e., we are given a set $R=\{t_1,\dots,t_k\}$ of k terminals and the set of demands is $\mathcal{D}=\{(t_i,t_{i+1})\}_{i=1}^{k}$ where $t_{k+1}=t_1$. Since this implies that any optimum solution is strongly connected, this problem is accordingly known as the STRONGLY CONNECTED STEINER SUBGRAPH (SCSS) problem. In contrast to DST, it is implicit from [31] (by a reduction from the CLIQUE problem) that optimum solutions to SCSS do not belong to any minor-closed graph class. Thus SCSS is not easily captured by some DSN$_{\mathcal{K}}$ problem for a restricted class $\mathcal{K}$. Nevertheless it is still possible to exploit the structure of the optimum solution to SCSS, which results in the following findings.

SCSS: It is known that a 2-approximation is obtainable [13] when parametrizing by k. We prove that the factor of 2 is best possible under Gap-ETH. To the best of our knowledge, this is the first example of a W[1]-hard problem having a parameterized approximation algorithm with non-trivial approximation factor (in this case 2), which is also known to be tight!

BI-SCSS, i.e., the SCSS problem on bidirected inputs: As for BI-DSN, one might think that BI-SCSS is easily solvable via its undirected version, i.e., the well-known STEINER TREE (ST) problem, which is FPT [53, 19] for parameter k. However, it is not the case that simply taking an optimum undirected solution twice in a bidirected graph will produce a (near-)optimum solution to BI-SCSS (see Figure 1). Nevertheless we prove that BI-SCSS is FPT for parameter k as well, while also being NP-hard. Our algorithm is non-trivial and does not apply any methods used for undirected graphs. To the best of our knowledge, bidirected inputs are the first example where SCSS remains NP-hard but turns out to be FPT parameterized by k! Thus in contrast to BI-DSN, the complexity of the in-between BI-SCSS problem resembles that of the undirected variant (the ST problem) rather than the directed version (the SCSS problem).

1.1 Our results

Due to space constraints, almost all proofs of the following theorems are deferred to the full version of the paper [12].

Bidirected inputs with planar solutions. Our main theorem implies the existence of a PAS for BI-DSN$_{\text{PLANAR}}$, where the parameter is the number k of demands.

▶ **Theorem 1.** *For any $\varepsilon>0$, there is a* $\max\left\{2^{k^{2^{O(1/\varepsilon)}}}, n^{2^{O(1/\varepsilon)}}\right\}$ *time algorithm for* BI-DSN$_{\text{PLANAR}}$*, that computes a $(1+\varepsilon)$-approximation.*

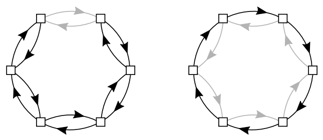


Figure 1 A BI-SCSS instance where all vertices are terminals. Left: Black edges show a solution which takes an undirected optimum twice. Right: The actual optimum solution is shown in black.

As BI-DSN$_{\text{PLANAR}}$ is a rather restricted special case of DSN, one may at this point rightfully ask: Should it not be possible to obtain better runtimes and/or should it not be possible to even compute the optimum solution when parametrizing by k? And could it not be that a similar result is true in more general settings, when for instance the input is bidirected but the optimum is not restricted to a planar graph? We prove that both questions can be answered in the negative.

First off, it is not hard to prove that a *polynomial time approximation scheme (PTAS)* is not possible for BI-DSN$_{\text{PLANAR}}$, i.e., it is necessary to parametrize by k in Theorem 1. This is implied by the following result, since (as mentioned before) a PTAS for BI-DSN$_{\text{PLANAR}}$ would also imply a PTAS for BI-DST, i.e., the DST problem on bidirected input graphs.

▶ **Theorem 2.** *The* BI-DST *problem is* APX-hard*.*

One may wonder however, whether parametrizing by k doesn't make the BI-DSN$_{\text{PLANAR}}$ problem FPT, so that approximating the planar optimum as in Theorem 1 would in fact be unnecessary. Furthermore, even if it is necessary to approximate, one may ask whether the runtime given in Theorem 1 can be improved. In particular, note that the runtime we obtain in Theorem 1 is similar to that of a PTAS, i.e., the exponent of n in the running time depends on ε. Ideally we would like an EPAS, which has a runtime of the form $f(k,\varepsilon)\cdot n^{O(1)}$, i.e., we would like to treat ε as a parameter as well. The following theorem shows that both approximating and runtime dependence on ε are in fact necessary in Theorem 1.

▶ **Theorem 3.** *The* BI-DSN$_{\text{PLANAR}}$ *problem is* W[1]*-hard parameterized by k. Moreover, under ETH, for any computable functions $f(k)$ and $f(k,\varepsilon)$, and parameters k and $\varepsilon > 0$, the* BI-DSN$_{\text{PLANAR}}$ *problem has no $f(k)\cdot n^{o(\sqrt{k})}$ time algorithm to compute the optimum solution, and has no $f(k,\varepsilon)\cdot n^{o(\sqrt{k})}$ time algorithm to compute a $(1+\varepsilon)$-approximation.*

It stands out that to compute optimum solutions, this theorem rules out runtimes for which the dependence of the exponent of n is $o(\sqrt{k})$, while for the general DSN problem, as mentioned above, the both necessary and sufficient dependence of the exponent is linear in k [24, 14]. Could it be that BI-DSN$_{\text{PLANAR}}$ is just as hard as DSN when computing optimum solutions? The answer is no, as the next theorem shows.

▶ **Theorem 4.** *There is a $2^{O(k^{3/2}\log k)}\cdot n^{O(\sqrt{k})}$ time algorithm to compute the optimum solution for* BI-DSN$_{\text{PLANAR}}$*.*

This result is an example of the so-called "square-root phenomenon": planarity often allows runtimes that improve the exponent by a square root factor in terms of the parameter when compared to the general case [28, 50, 40, 47, 41, 52, 55, 54, 51]. Interestingly though, Chitnis et al. [14] show that under ETH, no $f(k)\cdot n^{o(k)}$ time algorithm can compute the optimum solution to DSN$_{\text{PLANAR}}$. Thus assuming a bidirected input graph in Theorem 4 is necessary (under ETH) to obtain a factor of $O(\sqrt{k})$ in the exponent of n.

Bidirected inputs. Since in contrast to BI-DSN$_{\text{PLANAR}}$, the BI-DSN problem does not restrict the optimum solutions, one may wonder whether a parameterized approximation scheme as in Theorem 1 is possible for this more general case as well. We answer this in the negative by proving the following result, which implies that restricting the optima to planar graphs was necessary for Theorem 1.

▶ **Theorem 5.** *Under Gap-ETH, there exists a constant $\alpha > 1$ such that for any computable function $f(k)$ there is no $f(k) \cdot n^{O(1)}$ time algorithm that computes an α-approximation for* BI-DSN.

We leave open whether a similar inapproximability result can be obtained for the other obvious generalization of BI-DSN$_{\text{PLANAR}}$, in which the input graph is unrestricted but we need to compute the planar optimum, i.e., the DSN$_{\text{PLANAR}}$ problem. We conjecture that no approximation scheme exists for this problem either.

What approximation factors can be obtained for BI-DSN when parametrizing by k, given the lower bound of Theorem 5 on one hand, and the before-mentioned result [17] that rules out a $k^{1/4-o(1)}$-approximation for DSN in time parameterized by k on the other? It turns out that it is not too hard to obtain a constant approximation for BI-DSN, given the similarity of bidirected graphs to undirected graphs. In particular, relying on the fact that for the undirected version of DSN, i.e. the SF problem, there is a polynomial time 2-approximation algorithm [1], and an FPT algorithm based on [19], we obtain the following theorem, which is also in contrast to Theorem 2.

▶ **Theorem 6.** *The* BI-DSN *problem admits a 4-approximation in polynomial time, and a 2-approximation in $2^{O(k \log k)} \cdot n^{O(1)}$ time.*

Even if Theorem 5 in particular shows that BI-DSN cannot be FPT under Gap-ETH, it does not give a strong lower bound on the runtime dependence in the exponent of n. However using the weaker ETH assumption we can obtain such a lower bound, as the next theorem shows. Interestingly, the obtained lower bound implies that when aiming for optimum solutions, the restriction to bidirected inputs does not make DSN much easier than the general case, as also for BI-DSN the $n^{O(k)}$ time algorithm by [24] is essentially best possible. This is in contrast to the BI-DSN$_{\text{PLANAR}}$ problem where the square-root phenomenon takes effect as shown by Theorem 4.

▶ **Theorem 7.** *The* BI-DSN *problem is* W[1]*-hard parameterized by k. Moreover, under ETH there is no $f(k) \cdot n^{o(k/\log k)}$ time algorithm for* BI-DSN*, for any computable function $f(k)$.*

Thus when considering bidirected inputs, which lie between directed and undirected graphs, by Theorem 6 the complexity of the BI-DSN problem rather resembles the undirected variant (the SF problem) in terms of approximations, while by Theorem 7 it resembles the directed version (the DSN problem) in terms of FPT algorithms.

Strongly connected solutions. Just like the more general DSN problem, the SCSS problem is W[1]-hard [31] parameterized by k, and is also hard to approximate as no polynomial time $O(\log^{2-\varepsilon} n)$-approximation is possible [32], unless NP $\subseteq$ ZTIME($n^{\text{polylog}(n)}$). However it is possible to exploit the structure of the optimum to SCSS to obtain a 2-approximation algorithm parameterized by k, as observed by Chitnis et al. [13]. This is because any strongly connected graph is the union of two arborescences, and these form solutions to DST. The 2-approximation follows, since DST is FPT by the classic result of [19]. Thus in contrast to DSN, for SCSS it is possible to beat any approximation factor obtainable in polynomial time when parametrizing by k.

▶ **Theorem 8** ([13]). *The* SCSS *problem admits a* 2*-approximation in* $3^k \cdot n^{O(1)}$ *time.*

An obvious question now is whether the approximation ratio of this rather simple algorithm can be improved. Interestingly we are able to show that this is not the case. To the best of our knowledge, this is the first example of a W[1]-hard problem having a parameterized approximation algorithm with non-trivial approximation factor (in this case 2), which is also known to be tight!

▶ **Theorem 9.** *Under Gap-ETH, for any* $\varepsilon > 0$ *and any computable function* $f(k)$*, there is no* $f(k) \cdot n^{O(1)}$ *time algorithm that computes a* $(2-\varepsilon)$*-approximation for* SCSS*.*

Bidirected inputs with strongly connected solutions. In light of the above results for restricted cases of DSN, what can be said about restricted cases of SCSS? It is implicit in the work of Chitnis et al. [14] that $\text{SCSS}_{\text{PLANAR}}$, i.e., the problem of computing the optimum strongly connected planar optimum, can be solved in $2^{O(k \log k)} \cdot n^{O(\sqrt{k})}$ time, while under ETH no $f(k) \cdot n^{o(\sqrt{k})}$ time algorithm is possible. Hence $\text{SCSS}_{\text{PLANAR}}$ is slightly easier than $\text{DSN}_{\text{PLANAR}}$ where the exponent of n needs to be linear in k, as mentioned before. On the other hand, the BI-SCSS problem turns out to be a lot easier to solve than BI-DSN. This is implied by the next theorem, which stands in contrast to Theorem 5 and Theorem 7. In particular, the in-between BI-SCSS problem behaves more like the undirected ST problem than the directed SCSS problem.

▶ **Theorem 10.** *There is a* $2^{O(2^{k^2-k})} \cdot n^{O(1)}$ *time algorithm for* BI-SCSS*, i.e., it is FPT for parameter* k*.*

Could it be that BI-SCSS is even solvable in polynomial time? We prove that this is not the case, as it is NP-hard. To the best of our knowledge, the class of bidirected graphs is the first example where SCSS remains NP-hard but turns out to be FPT parameterized by k! Moreover, note that the above algorithm has a doubly exponential runtime in k^2. We conjecture that a single exponential runtime should suffice, and we also obtain a lower bound result of this form, even if we restrict the optimum solutions to very simple planar graphs, namely cycles.

▶ **Theorem 11.** *The* $\text{BI-SCSS}_{\text{CYCLE}}$ *problem is NP-hard. Moreover, under ETH there is no* $2^{o(k)} \cdot n^{O(1)}$ *time algorithm for* $\text{BI-SCSS}_{\text{CYCLE}}$*.*

▶ Remark. For ease of notation, throughout this paper we chose to use the number of demands k uniformly as the parameter. Alternatively one might also consider the smaller parameter $|R|$, where $R = \bigcup_{i=1}^{k}\{s_i, t_i\}$ is the set of terminals. Note for instance that in case of the SCSS problem, $k = |R|$, while for DSN, k can be as large as $\Theta(|R|^2)$ (cf. [22]). However we always have $k \geq |R|/2$, since the demands can form a matching in the worst case. It is interesting to note that all our algorithms for DSN have the same running time for parameter $|R|$ as for parameter k. That is, we may set $k = |R|$ in Theorem 1, 4, and 6.

1.2 Our techniques

It is already apparent from the above exposition of our results, that understanding the structure of the optimum solution is a powerful tool when studying DSN and its related problems (see Table 1). This is also apparent when reading the literature on these problems, and we draw some of our inspiration from these known results, as described below.

Table 1 Summary of achievable runtimes for DSN and SCSS when parameterizing by k. Some of the previous results are implicit and, in the papers, are rather stated for the case when the input graphs are restricted to the same class as the optimum solutions. Non-bracketed reference numbers refer to theorems of this paper.

	algorithms			lower bounds		
problem	approx.	runtime	ref.	approx.	runtime	ref.
DSN	–	$n^{O(k)}$	[24]	–	$f(k) \cdot n^{o(k)}$	[31]
DSN	$O(k^{\frac{1}{2}+\varepsilon})$	$n^{O(1)}$	[9]	$k^{\frac{1}{4}-o(1)}$	$f(k) \cdot n^{O(1)}$	[17]
DSN$_{\text{TW}:\ \omega}$	–	$2^{O(k\omega \log \omega)} \cdot n^{O(\omega)}$	[27]	–	$f(k, \omega) \cdot n^{o(\omega)}$	[27]
BI-DSN$_{\text{PLANAR}}$	$1+\varepsilon$	$\max\{2^{k^{2^{O(1/\varepsilon)}}}, n^{2^{O(1/\varepsilon)}}\}$	1	$1+\varepsilon$	$f(\varepsilon, k) \cdot n^{o(\sqrt{k})}$	3
BI-DSN$_{\text{PLANAR}}$	–	$2^{O(k^{3/2} \log k)} \cdot n^{O(\sqrt{k})}$	4	–	$f(k) \cdot n^{o(\sqrt{k})}$	3
DSN$_{\text{PLANAR}}$	–	$n^{O(k)}$	[24]	–	$f(k) \cdot n^{o(k)}$	[14]
BI-DSN	–	$n^{O(k)}$	[24]	–	$f(k) \cdot n^{o(k/\log k)}$	7
BI-DSN	2	$2^{O(k \log k)} \cdot n^{O(1)}$	6	$\alpha \in \Theta(1)$	$f(k) \cdot n^{O(1)}$	5
BI-DSN	4	$n^{O(1)}$	6	$\alpha \in \Theta(1)$	$n^{O(1)}$	2
SCSS	–	$n^{O(k)}$	[24]	–	$f(k) \cdot n^{o(k/\log k)}$	[14]
SCSS	2	$3^k \cdot n^{O(1)}$	[13]	$2-\varepsilon$	$f(k) \cdot n^{O(1)}$	9
SCSS$_{\text{PLANAR}}$	–	$2^{O(k)} \cdot n^{O(\sqrt{k})}$	[14]	–	$f(k) \cdot n^{o(\sqrt{k})}$	[14]
BI-SCSS	–	$2^{O(2^{k^2-k})} \cdot n^{O(1)}$	10	–	$2^{o(k)} \cdot n^{O(1)}$	11

For our approximation scheme for BI-DSN$_{\text{PLANAR}}$, we generalize the insights on the structure of optimum solutions to the classical STEINER TREE (ST) problem for our main result in Theorem 1. For the ST problem, an *undirected* edge-weighted graph is given together with a terminal set R, and the task is to compute the cheapest tree connecting all k terminals. For the ST problem only polynomial time 2-approximations were known [30, 60], until it was taken into account [36, 56, 63, 58] that any optimum Steiner tree can be decomposed into so-called *full components*, i.e., subtrees for which exactly the leaves are terminals. If a full component contains only a small subset of size k' of the terminals, it is the solution to an ST instance, for which the optimum can be computed efficiently in time $(2+\delta)^{k'} \cdot n^{O(1)}$ for any constant $\delta > 0$ using the algorithm of Mölle et al. [53]. A fundamental observation proved by Borchers and Du [6] is that for any k' there exists a solution to ST of cost at most $1 + \frac{1}{\lfloor \log_2 k' \rfloor}$ times the optimum, in which every full component contains at most k' terminals. Thus setting $k' = 2^{1/\varepsilon}$ for some constant $\varepsilon > 0$, all full-components with at most $2^{1/\varepsilon}$ terminals can be computed in polynomial time, and among them exists a collection forming a $(1+\varepsilon)$-approximation. The key to obtain approximation ratios smaller than 2 for ST is to cleverly select a good subset of all computed full-components. This is for instance done in [7] via an iterative rounding procedure, resulting in an approximation ratio of $\ln(4) + \varepsilon < 1.39$, which currently is the best one known.

Our main technical contribution is to generalize the Borchers-Du [6] Theorem to the BI-DSN$_{\text{PLANAR}}$ problem. In particular, to obtain our approximation scheme of Theorem 1, we employ a similar approach by decomposing a BI-DSN$_{\text{PLANAR}}$ solution into sub-instances, each containing a small number of terminals. As BI-DSN$_{\text{PLANAR}}$ is W[1]-hard by Theorem 3, we cannot hope to compute optimum solutions to each sub-instance as efficiently as for ST.

However, we provide an XP-algorithm with runtime $2^{O(k^{3/2}\log k)} \cdot n^{O(\sqrt{k})}$ for BI-DSN$_{\text{PLANAR}}$ in Theorem 4. Thus if every sub-instance contains at most $2^{1/\varepsilon}$ terminals, each can be solved in $n^{2^{O(1/\varepsilon)}}$ time, and this accounts for the "non-efficient" runtime of our approximation scheme. Since we allow runtimes parameterized by k, we can then exhaustively search for a good subset of precomputed small optimum solutions to obtain a solution to the given demand set $\mathcal{D}$. For the latter solution to be a $(1+\varepsilon)$-approximation however, we need to generalize the Borchers-Du [6] Theorem for ST to BI-DSN$_{\text{PLANAR}}$ (see Theorem 13 for the formal statement). This constitutes the bulk of the work to prove Theorem 1.

For our exact algorithms for BI-DSN$_{\text{PLANAR}}$ and BI-SCSS, we note that also from a parameterized point of view, understanding the structure of the optimum solution to DSN has lead to useful insights in the past. We will leverage one such recent result by Feldmann and Marx [27]. In [27] the above mentioned standard special case of restricting the patterns of the demands in $\mathcal{D}$ is studied in depth. The result is a complete dichotomy over which classes of restricted patterns define special cases of DSN that are FPT and which are W[1]-hard for parameter k. The high-level idea is that whenever the demand patterns imply optimum solutions of constant treewidth, there is an FPT algorithm computing such an optimum. In contrast, the problem is W[1]-hard whenever the demand patterns imply the existence of optimum solutions of arbitrarily large treewidth. The FPT algorithm from [27] lies at the heart of all our positive results, and therefore shows that the techniques developed in [27] to optimally solve special cases of DSN can be extended to find (near-)optimum solutions for other W[1]-hard special cases as well. It is important to note that the algorithm of [27] can also be used to compute the cheapest solution of treewidth at most ω, even if there is an even better solution of treewidth larger than ω (which might be hard to compute). Formally, the result leveraged in this paper is the following.

▶ **Theorem 12** (implicit in Theorem 5 of [27]). *If $\mathcal{K}$ is the class of graphs with treewidth at most ω, then the* DSN$_{\mathcal{K}}$ *problem can be solved in time $2^{O(k\omega\log\omega)} \cdot n^{O(\omega)}$.*

We exploit the algorithm given in Theorem 12 to prove our algorithmic results of Theorem 4 and Theorem 10. In particular, we prove that any BI-DSN$_{\text{PLANAR}}$ solution has treewidth $O(\sqrt{k})$, from which Theorem 4 follows immediately. For BI-SCSS however, we give an example of an optimum solution of treewidth $\Omega(k)$. Hence we cannot exploit the algorithm of Theorem 12 directly to obtain Theorem 10. In fact on general input graphs, a treewidth of $\Omega(k)$ would imply that the problem is W[1]-hard by the hardness results in [27] (which was indeed originally shown by Guo et al. [31]). As this stands in stark contrast to Theorem 10, it is particularly interesting that the problem on bidirected input graphs is FPT. We prove this result by decomposing an optimum solution to BI-SCSS into instances of BI-SCSS$_{\mathcal{K}}$, where $\mathcal{K}$ is the class of directed graphs of treewidth 1 (so-called *poly-trees*). For each such sub-instance we can compute a solution in $2^{O(k)} \cdot n^{O(1)}$ time by using Theorem 12 (for $\omega = 1$), and then combine them into an optimum solution to BI-SCSS.

Our hardness proofs for BI-DSN are based on reductions from the GRID TILING problem [15]. This problem is particularly suited to prove hardness for problems on planar graphs, due to its grid-like structure. We first develop a specific gadget that can be exploited to show hardness for bidirected graphs. This gadget however is not planar. We only exploit the structure of GRID TILING to show that the optimum solution is planar for Theorem 3. For Theorem 7 we modify this reduction to obtain a stronger runtime lower bound, but in the process we lose the property that the optimum is planar.

Our parameterized inapproximability result for SCSS is proved by combining a variant of a known reduction by Guo et al. [31] with a recent parameterized hardness of approximation result for Densest k-Subgraph [8]. Our inapproximability result for bi-DSN is shown by combining our W[1]-hardness reduction with the same hardness of approximation result of Densest k-Subgraph.

2 An approximation scheme for bi-DSN$_{\text{Planar}}$

In this section we prove Theorem 1. Note that since we have k demand pairs, it follows that the number of terminals $|R|$ is at most $2k$, where $R = \bigcup_{i=1}^{k}\{s_i, t_i\}$. Henceforth in this section, we use the upper bound $2k$ on the number of terminals $|R|$ for ease of presentation (when instead we could replace k by $|R|$ in the running time of Theorem 1). The bulk of the proof is captured by the following result, which generalizes the corresponding theorem by Borchers and Du [6] for the ST problem, and which is our main technical contribution. In order to facilitate the definition of a sub-instance to DSN, we encode the demands of a DSN instance using a *pattern graph* H, as also done in [27]: the vertex set of H is the terminal set R, and H contains the directed edge st if and only if (s,t) is a demand. Hence the DSN problem asks for a minimum cost network $N \subseteq G$ having an $s \to t$ path for each edge st of H.

▶ **Theorem 13.** *Let G be a bidirected graph, and H a pattern graph on $R \subseteq V(G)$. Let $N \subseteq G$ be an optimum* bi-DSN$_{\text{Planar}}$ *solution to H, i.e. N is planar. For any $\varepsilon > 0$, there exists a set of patterns $\mathcal{H}$ such that for each $H' \in \mathcal{H}$ there is a feasible* bi-DSN$_{\text{Planar}}$ *solution $N_{H'} \subseteq G$ and $|V(H')| \le 2^{O(1/\varepsilon)}$. Furthermore, the union $\bigcup_{H'\in\mathcal{H}} N_{H'}$ of the these solutions forms a feasible* bi-DSN$_{\text{Planar}}$ *solution to H with $\sum_{H'\in\mathcal{H}} \text{cost}(N_{H'}) \le (1+\varepsilon)\cdot\text{cost}(N)$.*

Based on Theorem 13 our $(1+\varepsilon)$-approximation algorithm proceeds as follows. The first step is to compute an optimum solution for every possible pattern graph on at most $g(\varepsilon) = 2^{O(1/\varepsilon)}$ terminals. Since any pattern graph has at most $2\binom{g(\varepsilon)}{2} < g(\varepsilon)^2$ edges, and there is a total of $2\binom{2k}{2} < 8k^2$ possible demands between the $2k$ terminals, the total number of pattern graphs is $O(k^{2g(\varepsilon)^2}) = k^{2^{O(1/\varepsilon)}}$. For each pattern the algorithm computes the optimum bi-DSN$_{\text{Planar}}$ solution in time $2^{g(\varepsilon)^{3/2}\log g(\varepsilon)} \cdot n^{O(\sqrt{g(\varepsilon)})} = n^{2^{O(1/\varepsilon)}}$ using the algorithm of Theorem 4. This amounts to a total runtime of $k^{2^{O(1/\varepsilon)}} \cdot n^{2^{O(1/\varepsilon)}}$ up to this point. The algorithm then proceeds by considering each subset $\mathcal{H}$ of the pattern graphs, and checking whether the union of the precomputed optimum solutions to all $H' \in \mathcal{H}$ forms a feasible solution to the input pattern H on R. As there are $2^{O(k^{2g(\varepsilon)^2})}$ subsets $\mathcal{H}$, and checking whether a subset induces a feasible solution can be done in polynomial time, this takes $2^{O(k^{2g(\varepsilon)^2})} \cdot n^{O(1)} = 2^{k^{2^{O(1/\varepsilon)}}} \cdot n^{O(1)}$ time. Among all feasible unions the algorithm outputs the solution with smallest cost. According to Theorem 13 this solution is a $(1+\varepsilon)$-approximation, and the total runtime is $k^{2^{O(1/\varepsilon)}} \cdot n^{2^{O(1/\varepsilon)}} + 2^{k^{2^{O(1/\varepsilon)}}} \cdot n^{O(1)} = \max\left\{2^{k^{2^{O(1/\varepsilon)}}}, n^{2^{O(1/\varepsilon)}}\right\}$. Thus we obtain Theorem 1.

Note that even though the output of the algorithm is a $(1+\varepsilon)$-approximation to the optimum bi-DSN$_{\text{Planar}}$ solution, the computed solution may not be planar, as it is the union of several planar graphs. Theorem 13 shows though that the structure of the optimum can be exploited to compute a near-optimum solution. We also note that the Borchers-Du[6] Theorem for the ST problem implies the existence of a *polynomial-sized* $(1+\varepsilon)$*-approximate kernel* for ST, as recently shown by Lokshtanov et al. [46]. By the same arguments this is also true for bi-DSN$_{\text{Planar}}$, due to Theorem 13. We refer to [46] for more details.

▶ **Corollary 14** (cf. [46]). *The* bi-DSN$_{\text{Planar}}$ *problem admits a polynomial-size approximate kernelization scheme (PSAKS) parameterized by k.*

It remains to prove Theorem 13. For this we assume we know the optimum planar solution $N \subseteq G$, and first use a standard transformation on N, so that each terminal has only 1 neighbour, each Steiner vertex has exactly 3 neighbours, and every pair of edges uv and vu have unique costs. Furthermore, let G_N be the graph spanned by the edge set $\{uv, vu \in E(G) \mid uv \in E(N)\}$, i.e. it is the underlying bidirected graph of N after performing the transformations on N. In particular, also in G_N each terminal has only 1 neighbour, each Steiner vertex has exactly 3 neighbours, and every pair of edges uv and vu have unique costs. It is not hard to see that proving Theorem 13 for the obtained optimum solution N in G_N implies the same result for the original optimum solution in G, by reversing all transformations.

The proof consists of two parts, of which the first exploits the bidirectedness of G_N, while the second exploits that the optimum N is planar. The first part will identify paths connecting each Steiner vertex to some terminal in such a way that the paths do not overlap much. This will enable us to select a subset of these paths in the second part, so that the total weight of the selected paths is an ε-fraction of the cost of the optimum solution. This subset of paths will be used to connect terminals to the boundary vertices of small regions into which we divide the optimum. These regions extended by the paths then form solutions to sub-instances to DSN, which together have a cost of $1+\varepsilon$ times the optimum. The first part is captured by the next lemma, where $\text{cost}(G')$ denotes the total edge weight of a graph G'.

▶ **Lemma 15.** *Let $N \subseteq G_N$ be the optimum* BI-DSN$_{\text{PLANAR}}$ *solution to a pattern graph H on $R \subseteq V(G_N)$. For every Steiner vertex $v \in V(N) \setminus R$ of N there is a path P_v in G_N, such that P_v is a $v \to t$ path to some terminal $t \in R$, and the total cost $\sum_{v \in V(N)\setminus R} \text{cost}(P_v)$ of these paths is $O(\text{cost}(N))$.*

For the second part we give each vertex v of N a weight $c(v)$, which is zero for terminals and equal to $\text{cost}(P_v)$ for each Steiner vertex $v \in V(N) \setminus R$ and corresponding path P_v given by Lemma 15. We now divide the optimum solution N into regions of small size, such that the boundaries of the regions have small total weight. Formally, a *region* is a subgraph of N, and an *r-division* is given by a partition of the edges of N, each spanning a region with at most r vertices. A *boundary vertex* of an r-division is a vertex that lies in at least two regions. In a *weak r-division*, as for instance defined in [35], we bound the total number of boundary vertices and the number of regions (it is called "weak" since it does not bound the boundary vertices of each region individually). For unweighted planar graphs it can be shown that there is an r-division with only $O(n/\sqrt{r})$ boundary vertices and $O(n/r)$ regions [35, 29]. To prove this, a separator theorem is applied recursively until each resulting region is small enough. The bound on the number of boundary vertices follows from the well-known fact that any planar graph has a small separator of size $O(\sqrt{n})$.

We however need to bound the total weight of the boundary vertices, i.e. we need a *weighted weak r-division*. Unfortunately, separator theorems are not helpful here, since they only bound the number of vertices in the separator but cannot bound their weight. Instead we leverage techniques developed for the Klein-Plotkin-Rao (KPR) Theorem [45, 39] in order to show that there is an r-division for which the total weight of all boundary vertices is an $O(1/\log r)$-fraction of the total weight $\sum_{v \in V(N)} c(v)$, if the graph has constant maximum degree. We later set $r = 2^{1/\varepsilon}$ in order to obtain an ε-fraction of the total weight. Even though the obtained fraction is exponentially worse than the $O(1/\sqrt{r})$-fraction for unweighted graphs obtained in [35, 29], it follows from a lower bound result of Borchers and Du [6] that for weighted graphs this is best possible, even if the graph is a tree. In contrast to the unweighted case, we also do not guarantee any bound on the number of regions, and we do not need such a bound either. Our proof follows the outlines of the proof given by Lee [45] for the KPR Theorem. In the following, $c(S) = \sum_{v \in S} c(v)$ for any set of vertices S.

▶ **Lemma 16.** *Let N be a directed planar graph for which each vertex has at most 3 neighbours, and let each vertex v of N have a weight $c(v) \in \mathbb{R}$. For any $r \in \mathbb{N}$ there is a partition $\mathcal{E}$ of the edges of N for which every set in $\mathcal{E}$ spans at most r vertices, and if B is the set of boundary vertices of the regions spanned by the sets in $\mathcal{E}$, then $c(B) = O\left(\frac{c(V(N))}{\log r}\right)$.*

We here only prove some parts of Lemma 15 (cf. [12] for the full version of the paper).

Proof of Lemma 15. We begin by analysing the structure of optimal DSN solutions in bidirected graphs. Here a *condensation graph* of a directed graph results from contracting each strongly connected component, which hence is a DAG. A *poly-forest* is obtained by directing the edges of an undirected forest.

▶ **Claim 17.** *For any solution $N \subseteq G_N$ to a pattern H, there is a solution $M \subseteq G_N$ to H with $\operatorname{cost}(M) \leq \operatorname{cost}(N)$, such that the condensation graph of M is a poly-forest.*

By Claim 17 we may assume w.l.o.g. that the condensation graph of the optimum solution N is a poly-forest. Consider a weakly connected component C of N, i.e. inducing a connected component of the underlying undirected graph $\overleftrightarrow{N}$. We first extend C to a strongly connected graph C' as follows. Let F be the edges of C that do not lie in a strongly connected component, i.e. they are the edges of the condensation graph of C. Let $\widetilde{F} = \{uv \mid vu \in F\}$ be the set containing the reverse edges of F, and let C' be the strongly connected graph spanned by all edges of C in addition to the edges in $\widetilde{F}$. Note that adding $\widetilde{F}$ to C increases the cost by at most a factor of two as G_N is bidirected, and the number of neighbours of any vertex does not change. We claim that in fact C' is a *minimal* SCSS solution to the terminal set $R_C \subseteq R$ contained in C, that is, removing any edge of C' will disconnect some terminal pair of R_C.

For this, consider any $s \to t$ path of C' containing an edge $e \in \widetilde{F}$ for some terminal pair $s, t \in R_C$. As the edges F of the condensation graph of C form a poly-tree, every path from s to t in C' must pass through e. In particular there is no $s \to t$ path in C, and thus there is no edge st in the pattern graph H. Or conversely, for any terminal pair $s, t \in R_C$ for which there is a demand $st \in E(H)$, no $s \to t$ path in C' passes through an edge of $\widetilde{F}$. Thus the set of paths from s to t is the same in C' and C. Since every edge e of the weakly connected component C is necessary for some such pair $s, t \in R_C$ with $st \in E(H)$, the edge e is still necessary in C'. Moreover, for any of the added edges $uv \in \widetilde{F}$ the reverse edge $vu \in F$ was necessary in C to connect some $s \in R_C$ to some $t \in R_C$. As observed above, uv is necessary to connect t to s in C', since the edges F of the condensation graph form a poly-tree.

As C' is a minimal SCSS solution to the terminals R_C contained within, it is the union of an in-arborescence A_{in} and out-arborescence A_{out}, both with the same root $r \in R_C$ and leaf set $R_C \setminus \{r\}$, since every terminal only has one neighbour in G_N. A *branching point* of an arborescence A is a vertex with at least two children in A. We let $W \subseteq V(C')$ be the set consisting of all terminals R_C and all branching points of A_{in} and A_{out}. We will need that any vertex of C' has a vertex of W in its close vicinity. That is, if $\Delta[v] = \{u \in V(C') \mid u = v \vee uv \in E(C') \vee vu \in E(C')\}$ denotes the inclusive neighbourhood of a vertex v ignoring directions of edges and $\Delta^2[v] = \bigcup_{u \in \Delta[v]} \Delta[v]$, we prove the following.

▶ **Claim 18.** *For every vertex v of C', there is a vertex of W in $\Delta^2[v]$.*

As the graph G_N is bidirected, for any v-u path P in the underlying undirected graph $\overleftrightarrow{G}_N$ of G_N, there exists a corresponding directed $v \to u$ path in G_N of the same cost. Therefore, we can ignore the directions of the edges in C' and the arborescences A_{out} and A_{in} to identify the paths P_v for Steiner vertices v of N. Thus we will only consider paths in the underlying undirected graphs $\overleftrightarrow{C}'$, $\overleftrightarrow{A}_{out}$, and $\overleftrightarrow{A}_{in}$ from now on. In particular, we exploit the following observation found in [20] (and also used by [6]) on undirected trees.

▶ **Claim 19** ([20, Lemma 3.2]). *For any undirected tree T we can find a path $P_v \subseteq T$ for every branching point v, such that P_v leads from v to some leaf of T, and all these paths P_v are pairwise edge-disjoint.*

If a Steiner vertex v of C' is a branching point of A_{out} (A_{in}), we let P_v be the corresponding path in $\overleftrightarrow{A}_{out}$ ($\overleftrightarrow{A}_{in}$) given by Claim 19 from v to some leaf of A_{out} (A_{in}), which is a terminal. Note that paths in $\overleftrightarrow{A}_{in}$ may overlap with paths in $\overleftrightarrow{A}_{out}$. However any edge in the union of all the paths P_v chosen so far is contained in at most two such paths, one for a branching point of A_{out} and one for a branching point of A_{in}.

It remains to choose a path P_v for every Steiner vertex v that is neither a branching point of A_{out} nor of A_{in}, i.e. for every vertex not in W. By Claim 18 for any such vertex $v \notin W$ there is a vertex $u \in \Delta^2[v]$ for which $u \in W$. If u is a terminal, then the path P_v is simply the edge vu if $u \in \Delta[v]$ or the corresponding path vwu for some $w \in \Delta[v]$ otherwise. If u is not a terminal but a branching point of A_{out} or A_{in}, then we chose a path P_u for u above. In this case, P_v is the path contained in the walk given by extending the path P_u by the edge vu or the path vwu, respectively. Note that, as any vertex of C' has at most three neighbours, any terminal or branching point $u \in W$ can be used in this way for some vertex $v \notin W$ at most nine times. Therefore any edge in the union of all chosen paths is contained in $O(1)$ paths. Consequently the total cost $\sum_{v\in V(N)\setminus R} \text{cost}(P_v)$ is $O(\text{cost}(C'))$, and as $\text{cost}(C') \le 2\,\text{cost}(C)$ we also get $\sum_{v\in V(N)\setminus R} \text{cost}(P_v) = O(\text{cost}(C))$.

We may repeat these arguments for every weakly connected component of N to obtain the lemma. ◀

References

1 Ajit Agrawal, Philip Klein, and R Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456, 1995.

2 Benny Applebaum. Exponentially-Hard Gap-CSP and Local PRG via Local Hardcore Functions. In *FOCS 2017*, pages 836–847, 2017. `doi:10.1109/FOCS.2017.82`.

3 Piotr Berman, Arnab Bhattacharyya, Konstantin Makarychev, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Approximation algorithms for spanner problems and directed steiner forest. *Information and Computation*, 222:93–107, 2013.

4 Arnab Bhattacharyya, Suprovat Ghoshal, Karthik C. S., and Pasin Manurangsi. Parameterized Intractability of Even Set and Shortest Vector Problem from Gap-ETH. *To appear in ICALP 2018*, 2018. `arXiv:1803.09717`.

5 Edouard Bonnet, Bruno Escoffier, EunJung Kim, and Vangelis T. Paschos. On subexponential and FPT-time inapproximability. In *IPEC*, pages 54–65, 2013.

6 Al Borchers and Ding-Zhu Du. The k-Steiner Ratio in Graphs. *SIAM Journal on Computing*, 26(3):857–869, 1997.

7 Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *Journal of the ACM*, 60(1):6, 2013.

8 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From Gap-ETH to FPT-Inapproximability: Clique, Dominating Set, and More. In *To appear in FOCS*, 2017.

9 Chandra Chekuri, Guy Even, Anupam Gupta, and Danny Segev. Set connectivity problems in undirected graphs and the directed Steiner network problem. *ACM Transactions on Algorithms*, 7(2):18, 2011.

10 W-T Chen and N-F Huang. The strongly connecting problem on multihop packet radio networks. *IEEE Transactions on Communications*, 37(3):293–295, 1989.

11 Yijia Chen and Bingkai Lin. The constant inapproximability of the parameterized dominating set problem. In *FOCS*, pages 505–514, 2016.

12 Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi. Parameterized Approximation Algorithms for Directed Steiner Network Problems. *CoRR*, abs/1707.06499, 2017. arXiv:1707.06499.

13 Rajesh Chitnis, MohammadTaghi Hajiaghayi, and Guy Kortsarz. Fixed-parameter and approximation algorithms: A new look. In *IPEC*, pages 110–122, 2013.

14 Rajesh Chitnis, MohammadTaghi Hajiaghayi, and Dániel Marx. Tight bounds for planar strongly connected Steiner subgraph with fixed number of terminals (and extensions). In *SODA*, pages 1782–1801, 2014.

15 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

16 Irit Dinur. Mildly exponential reduction from gap 3SAT to polynomial-gap label-cover. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:128, 2016.

17 Irit Dinur and Pasin Manurangsi. ETH-Hardness of Approximating 2-CSPs and Directed Steiner Network. In *ITCS*, pages 36:1–36:20, 2018. doi:10.4230/LIPIcs.ITCS.2018.36.

18 Yevgeniy Dodis and Sanjeev Khanna. Design networks with bounded pairwise distance. In *STOC 1999*, pages 750–759, 1999.

19 S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

20 Ding-Zhu Du, Yanjun Zhang, and Qing Feng. On better heuristic for Euclidean Steiner minimum trees. In *FOCS 1991*, pages 431–439, 1991.

21 Pavel Dvorák, Andreas Emil Feldmann, Dusan Knop, Tomás Masarík, Tomas Toufar, and Pavel Veselý. Parameterized approximation schemes for steiner trees with small number of steiner vertices. In *STACS*, pages 26:1–26:15, 2018.

22 Eduard Eiben, Dušan Knop, Fahad Panolan, and Ondřej Suchý. Complexity of the steiner network problem with respect to the number of terminals. *arXiv preprint*, 2018. arXiv:1802.08189.

23 Eduard Eiben, Mithilesh Kumar, Amer E Mouawad, and Fahad Panolan. Lossy kernels for connected dominating set on sparse graphs. *arXiv preprint*, 2017. arXiv:1706.09339.

24 Jon Feldman and Matthias Ruhl. The directed Steiner network problem is tractable for a constant number of terminals. *SIAM J. Comput.*, 36(2):543–561, 2006.

25 Moran Feldman, Guy Kortsarz, and Zeev Nutov. Improved approximation algorithms for directed steiner forest. *J. Comput. Syst. Sci.*, 78(1):279–292, 2012. doi:10.1016/j.jcss.2011.05.009.

26 Andreas Emil Feldmann. Fixed parameter approximations for k-center problems in low highway dimension graphs. In *ICALP*, pages 588–600, 2015. doi:10.1007/978-3-662-47666-6_47.

27 Andreas Emil Feldmann and Dániel Marx. The complexity landscape of fixed-parameter directed steiner network problems. *CoRR*, abs/1707.06808, 2017. arXiv:1707.06808.

28 Fedor V. Fomin, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Subexponential Parameterized Algorithms for Planar and Apex-Minor-Free Graphs via Low Treewidth Pattern Covering. In *FOCS*, pages 515–524, 2016. doi:10.1109/FOCS.2016.62.

29 Greg N Frederickson and Joseph Ja'Ja'. Approximation algorithms for several graph augmentation problems. *SIAM Journal on Computing*, 10(2):270–283, 1981.

30 E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

31 Jiong Guo, Rolf Niedermeier, and Ondrej Suchý. Parameterized complexity of arc-weighted directed Steiner problems. *SIAM J. Discrete Math.*, 25(2):583–599, 2011.

32 Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In *STOC*, pages 585–594, 2003.

33 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

34 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

35 Giuseppe F Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *STOC 2011*, pages 313–322, 2011.

36 Marek Karpinski and Alexander Zelikovsky. New approximation algorithms for the Steiner tree problem. *Journal of Combinatorial Optimization*, 1(1):47–65, 1997.

37 Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the Parameterized Complexity of Approximating Dominating Set. *To appear in STOC 2018*, 2017. arXiv:1711.11029.

38 Hervé Kerivin and A Ridha Mahjoub. Design of survivable networks: A survey. *Networks*, 46(1):1–21, 2005.

39 Philip Klein, Serge A. Plotkin, and Satish Rao. Excluded Minors, Network Decomposition, and Multicommodity Flow. In *STOC 1993*, pages 682–690, 1993.

40 Philip N. Klein and Dániel Marx. Solving Planar k -Terminal Cut in $O(n^{c\sqrt{k}})$ Time. In *ICALP*, pages 569–580, 2012. doi:10.1007/978-3-642-31594-7_48.

41 Philip N. Klein and Dániel Marx. A subexponential parameterized algorithm for Subset TSP on planar graphs. In *SODA*, pages 1812–1830, 2014. doi:10.1137/1.9781611973402.131.

42 R. Krithika, Pranabendu Misra, Ashutosh Rai, and Prafullkumar Tale. Lossy kernels for graph contraction problems. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016)*, volume 65, pages 23:1–23:14, Dagstuhl, Germany, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.23.

43 Nhat X Lam, Trac N Nguyen, Min Kyung An, and Dung T Huynh. Dual power assignment optimization and fault tolerance in WSNs. *Journal of Combinatorial Optimization*, 30(1):120–138, 2015.

44 Michael Lampis. Parameterized approximation schemes using graph widths. In *ICALP*, pages 775–786, 2014.

45 James Lee. A simpler proof of the KPR theorem, 2012. URL: https://tcsmath.org/2012/01/11/a-simpler-proof-of-the-kpr-theorem/.

46 Daniel Lokshtanov, Fahad Panolan, MS Ramanujan, and Saket Saurabh. Lossy Kernelization. In *STOC*, pages 224–237, 2017.

47 Daniel Lokshtanov, Saket Saurabh, and Magnus Wahlström. Subexponential Parameterized Odd Cycle Transversal on Planar Graphs. In *FSTTCS*, pages 424–434, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.424.

48 Pasin Manurangsi and Prasad Raghavendra. A Birthday Repetition Theorem and Complexity of Approximating Dense CSPs. In *ICALP*, pages 78:1–78:15, 2017.

49 Dániel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.

50 Dániel Marx. A Tight Lower Bound for Planar Multiway Cut with Fixed Number of Terminals. In *ICALP*, pages 677–688, 2012. doi:10.1007/978-3-642-31594-7_57.

51 Dániel Marx, Marcin Pilipczuk, and Michał Pilipczuk. On subexponential parameterized algorithms for Steiner Tree and Directed Subset TSP on planar graphs. *arXiv preprint arXiv:1707.1707.02190*, 2017.

52 Dániel Marx and Michal Pilipczuk. Optimal Parameterized Algorithms for Planar Facility Location Problems Using Voronoi Diagrams. In *ESA*, pages 865–877, 2015. doi:10.1007/978-3-662-48350-3_72.

53 Daniel Mölle, Stefan Richter, and Peter Rossmanith. A faster algorithm for the steiner tree problem. In *STACS*, pages 561–570, 2006.

54 Marcin Pilipczuk, Michal Pilipczuk, Piotr Sankowski, and Erik Jan van Leeuwen. Subexponential-Time Parameterized Algorithm for Steiner Tree on Planar Graphs. In *STACS*, pages 353–364, 2013. doi:10.4230/LIPIcs.STACS.2013.353.

55 Marcin Pilipczuk, Michal Pilipczuk, Piotr Sankowski, and Erik Jan van Leeuwen. Network Sparsification for Steiner Problems on Planar and Bounded-Genus Graphs. In *FOCS*, pages 276–285, 2014. doi:10.1109/FOCS.2014.37.

56 Hans Jürgen Prömel and Angelika Steger. A new approximation algorithm for the Steiner tree problem with performance ratio 5/3. *Journal of Algorithms*, 36:89–101, 2000.

57 Ram Ramanathan and Regina Rosales-Hain. Topology control of multihop wireless networks using transmit power adjustment. In *INFOCOM*, volume 2, pages 404–413, 2000.

58 Gabriel Robins and Alexander Zelikovsky. Tighter bounds for graph Steiner tree approximation. *SIAM Journal on Discrete Mathematics*, 19(1):122–134, 2005.

59 Sebastian Siebertz. Lossy kernels for connected distance-r domination on nowhere dense graph classes. *arXiv preprint*, 2017. arXiv:1707.09819.

60 Vijay Virkumar Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

61 Chen Wang, Myung-Ah Park, James Willson, Yongxi Cheng, Andras Farago, and Weili Wu. On approximate optimal dual power assignment for biconnectivity and edge-biconnectivity. *Theoretical Computer Science*, 396(1-3):180–190, 2008.

62 Andreas Wiese. A $(1+\epsilon)$-approximation for unsplittable flow on a path in fixed-parameter running time. In *ICALP 2017*, pages 67:1–67:13, 2017.

63 Alexander Zelikovsky. An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica*, 9:463–470, 1993.

Online Facility Location with Deletions

Marek Cygan
Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland
cygan@mimuw.edu.pl

Artur Czumaj
Department of Computer Science and Centre for Discrete Mathematics and its Applications (DIMAP), University of Warwick, Coventry CV4 7AL, United Kingdom
A.Czumaj@warwick.ac.uk

Marcin Mucha
Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland
mucha@mimuw.edu.pl

Piotr Sankowski
Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland
sank@mimuw.edu.pl

Abstract

In this paper we study three previously unstudied variants of the online Facility Location problem, considering an intrinsic scenario when the clients and facilities are not only allowed to arrive to the system, but they can also depart at any moment.

We begin with the study of a natural *fully-dynamic online uncapacitated* model where clients can be both added and removed. When a client arrives, then it has to be assigned either to an existing facility or to a new facility opened at the client's location. However, when a client who has been also one of the open facilities is to be removed, then our model has to allow to reconnect all clients that have been connected to that removed facility. In this model, we present an optimal $O(\log \mathfrak{n}_{\mathfrak{act}}/\log\log \mathfrak{n}_{\mathfrak{act}})$-competitive algorithm, where $\mathfrak{n}_{\mathfrak{act}}$ is the number of active clients at the end of the input sequence.

Next, we turn our attention to the *capacitated* Facility Location problem. We first note that if no deletions are allowed, then one can achieve an optimal competitive ratio of $O(\log \mathfrak{n}/\log\log \mathfrak{n})$, where $\mathfrak{n}$ is the length of the sequence. However, when deletions are allowed, the capacitated version of the problem is significantly more challenging than the uncapacitated one. We show that still, using a more sophisticated algorithmic approach, one can obtain an online $O(\log \mathfrak{m} + \log \mathfrak{c} \log \mathfrak{n})$-competitive algorithm for the capacitated Facility Location problem in the fully dynamic model, where $\mathfrak{m}$ is number of points in the input metric and $\mathfrak{c}$ is the capacity of any open facility.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases online algorithms, facility location, fully-dynamic online algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.21

Related Version A full version of the paper is available at `https://arxiv.org/abs/1807.03839`.

Funding Research partially supported by the Royal Society International Exchanges Scheme 2013/R1, grant IE130346, by the Centre for Discrete Mathematics and its Applications (DIMAP), by EPSRC awards EP/D063191/1, EP/N011163/1, and by the European Research Council research and innovation programme (ERC grant agreement No 677651).

© Marek Cygan, Artur Czumaj, Marcin Mucha, and Piotr Sankowski;
licensed under Creative Commons License CC-BY
26th Annual European Symposium on Algorithms (ESA 2018).
Editors: Yossi Azar, Hannah Bast, and Grzegorz Herman; Article No. 21; pp. 21:1–21:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The Facility Location is one of the central combinatorial optimization problem, extensively studied in the literature for several decades, see, e.g., [5, 12, 14, 29, 37, 45] and the references therein. The goal is to connect a given set of clients to a set of facilities such that the service cost is optimized. A very natural setting for the Facility Location problem is the *online* scenario, where clients arrive incrementally over time and need to be connected to existing or newly opened facilities. Indeed, since it has been introduced over 15 years ago by Meyerson [40], the online version of the Facility Location problem and its generalizations received considerable amount of attention [3, 6, 7, 16, 17, 20, 21, 22, 25, 26, 33, 41, 42, 43, 48]. Typically in these models one assumes to be given a metric over a set of candidate points. When a client appears, we are allowed to open a new facility in her location, paying some cost for opening a new facility, and then we have to irrevocably connect the client to one of the open facilities, paying for the connection the cost equal to the distance from the client to the facility. In this paper, we study a more complex scenario and consider a natural extension of this model allowing clients to be *removed* from the system, extending the standard dynamic model to the *fully dynamic* setting. That is, in each time step either a new client arrives and needs to be connected to one of the open facilities, or one of the clients already existing in the system departs. Observe that if a client who has been also one of the open facilities is to be removed, then our model has to allow to reconnect all clients that have been connected to that removed facility.

The fully dynamic model is very natural in the context of the Facility Location problem, where in a number of scenarios it is desirable to dynamically process the arrival objects, and then to allow their departures. For example, if one wants to build and maintain schools in a newly developing city, one wants to allow a steady arrival of new pupils to the area, and also allow changes in the school demands when pupils population is declining. Similarly, if one wants to maintain a construction of a network, where all clients are to be connected to the servers (and pay connection costs) and each client is allowed to host a server (and pay an opening cost), one may also want to allow the removal of some clients and with that also a closure of some servers. Given that the relocation costs are often very expensive, the decisions should be regarded as irrevocable, unless, as in the case of deletions, are necessary. The framework considered in this paper is even more natural in complex distributed settings, for example, in some more modern scenarios that have been recently motivated by applications in peer-to-peer systems, when client and facility are coupled to the same entity. Consider the so called p2p networks with super-peers [49] that were used for filesharing as Gnutella [30] or can be used for decentralized online social networks [44], distributed game systems [47], grid management systems [19] and distributed storage [34]. In such case, some of the nodes decide to host the provided content. This decision incurs some cost to this node, but can reduce the cost of serving other nodes. The main question asked about such systems is about the needed ratio of superpeers to peers that guarantees that enough capacity of superpeers is available to serve all clients [49]. This measure corresponds exactly to competitive ratio in our models.

Before discussing our results and techniques further, let us first formally define the models starting with the classical online variant.

1.1 The model

We study the performance of online algorithms for the Facility Location problem in the standard framework of *competitive analysis* (cf. [11, 46]). A randomized algorithm for the Facility Location problem is *α-competitive* if for any input sequence, its expected cost is at most α times the optimal cost for the corresponding instance of the offline Facility Location. Note that the corresponding instance of the offline problem contains only the clients that are active at the end of the input sequence. We consider a standard special version of the Facility Location problem, where the set of clients and the set of possible facility locations are identical (see, e.g., [8, 23, 40]).

1.1.1 *Online* facility location

We consider the Facility Location problem, where points (from some metric space) arrive in online fashion. When a point x arrives, we can first open a facility in x and then we have to assign x to some open facility. These choices are irrevocable. We consider only the Facility Location problem with *uniform opening costs*, which are all, without loss of generality, equal to 1. The restriction to the study of uniform costs makes the problem interesting, since without this assumption, no bounded competitive ratio can be obtained (as described in the full version of this paper).

The cost of an obtained solution for a point set $\mathcal{X}$ is:

$$|\mathcal{F}| + \sum_{x \in \mathcal{X}} dist(x, \mathfrak{a}(x)) \ ,$$

where $\mathcal{F}$ is the set of open facilities and $\mathfrak{a}(x)$ is the open facility to which x is assigned. The cost of a point $x \in S$ is equal to its *connecting cost* (distance to the open facility assigned to x) plus its *opening cost* (if x is an open facility then the opening cost is 1, and it is 0 otherwise).

The model defined here has been studied in the past, see e.g., Fotakis [23] for a survey or Meyerson [40]. (One frequently assumes (see e.g., [40]) that the opening cost in any facility is equal to some f, but simple scaling makes this problem equivalent to the one studied in our paper, that is, with $f = 1$.)

1.1.2 Online facility location with *deletions*

In this paper, we consider a *fully-dynamic online* setting, in which in each time step either a new point (from some metric space) arrives in an online fashion, or a point already in the input is deleted. To cope with the case when an open facility is deleted, we have to allow the input points to be reassigned, and possibly to open other facilities. We consider the following, very natural model:

- In the online process, the requests are arriving online, and each request is either an arrival of a new demand point from a metric space, or a request to remove a previously inserted demand point.
- When a new demand point x arrives, the algorithm must at once and irrevocably decide if a new facility will open at x, and then must assign x to some open facility (possibly to itself, if a facility was open at x).
- When a point x is requested to be removed, then x is deleted from the system and if x was an open facility, then all points assigned to x will be immediately reassigned to other facilities and some of these points may become open facilities.

1.1.3 Online *capacitated* facility location

We also consider a more general model of online *capacitated* Facility Location with deletions, in which each open facility can handle at most $\mathfrak{c}$ clients, that is, at any moment at most $\mathfrak{c}$ clients can be connected to any single facility. Again, we study only a *uniform* case (in which each facility has the same capacity $\mathfrak{c}$), since otherwise no bounded competitive ratio can be obtained (see full version of the paper).

The goal of an algorithm for the Facility Location in any of the models defined above is to minimize the cost of the solution and to obtain an algorithm that is α-competitive for α as small as possible. The performance of the algorithm may be a function of the *length of the input sequence* $\mathfrak{n}$, the *number of points* $\mathfrak{n}_{\mathfrak{act}}$ *active* in a given moment, the *size of the input metric space* $\mathfrak{m}$, and the *capacity* $\mathfrak{c}$.

We note that in the model of uncapacitated Facility Location, the known results (as well as our new results) work for any metric space, even if it is *unknown to the algorithm.* However, for the new algorithm for online capacitated Facility Location with deletions (cf. Theorem 4.4), we will assume that *the underlying metric is known to the algorithm in advance.*

1.2 Related work

As one of the fundamental problems in operations research and combinatorial optimization, the Facility Location problem has been studied extensively in the past, see, e.g., the standard exposition in [14] and more recent advances in [4, 5, 29, 45], and the references therein. In the online setting, an early research focused on the k-median problem (see, e.g., [39]), which is a variant of the Facility Location problem where exactly k facilities have to be opened. Soon after, Meyerson [40] designed a simple randomized online algorithm in the uncapacitated model without deletion. His online algorithm is $O(\log \mathfrak{n}/\log\log \mathfrak{n})$-competitive, where $\mathfrak{n}$ is the number of points in the input. (In fact, only a competitive ratio of $O(\log \mathfrak{n})$ was proven in [40], but Fotakis [22] extended the analysis from [40] to obtain a competitive ratio of $O(\log \mathfrak{n}/\log\log \mathfrak{n})$.) Fotakis [22] later has shown that this bound is asymptotically tight and no online algorithm is $o(\log \mathfrak{n}/\log\log \mathfrak{n})$-competitive; the lower bound holds for randomized algorithms against the oblivious adversary, for uniform facility costs, and for very simple metric spaces, such as the real line. For more discussion about the history of the online version of the uncapacitated Facility Location problem (including deterministic online algorithms and incremental online algorithms), we refer to a survey by Fotakis [23].

The extension of the online model to deal with deletion of the facilities makes the Facility Location problem significantly more challenging. While this model is very natural, it requires a different approach that must permit to *reverse some* of the decisions in the online algorithm, and we are not aware of any study of algorithms for the online Facility Location problem in the fully dynamic setting, where deletions are allowed.

We note however, that similar fully dynamic models for optimization problems on graphs have been considered in the past, though only in limited settings. For example, a number of graph optimization problems have been considered in fully dynamic models in the setting of *data streams*, in the so-called *turnstile model.* This model has been investigated in two scenarios: in the context of geometric graph optimization problems (see, e.g., [15, 28, 35]), and only very recently, in the context of standard graph optimization problems (see, e.g., a recent survey [38] and the references therein). The main focus of these studies is to design algorithms that process a stream of data (in this case, edge or vertex insertions and deletions) and *using very limited space*, to maintain some basic graph features. For *geometric graph optimization problems*, where the input is defined over a set of points in the discrete d-dimensional space $\{1, 2, \dots, \Delta\}^d$, it has been shown that many basic properties (e.g., k-median, minimum spanning tree, minimum weight matching, MaxCut) can be approximated very efficiently even with poly-logarithmic space (see, e.g., [28, 24]). The uncapacitated Facility Location problem has been studied in the context of data streaming, initiated with work of Indyk [28], who gave a poly($\log \Delta$)-space streaming algorithm that approximates the optimal cost of the Facility Location problem within a factor of $O(\log^2 \Delta)$. The best currently known streaming algorithm using poly($\log \Delta$)-space gives an $(1+\varepsilon)$-approximation for this problem [15]. The research in data streaming for *standard graph optimization problems* has been traditionally focusing on the insertion-only model, where one was aiming to design streaming algorithms with $O(n \operatorname{poly} \log n)$ space (cf. [38] and the references therein). Only very recently, Ahn et al. [1] initiated the study of algorithms that allow both insertions and deletions (see also [13]). This line of research led to a number of efficient data streaming algorithms for fundamental graph problems, such as testing connectivity or bipartiteness, computing spanning trees and various graph sparsifiers, maximum matching, that can be (approximately) computed in small, $O(n \operatorname{poly} \log n)$ space not only in the insertion only model, but also in the model with deletions [1, 2, 31, 32, 38].

We also note that recently there has been some research on the standard online Steiner tree problem with deletions, see e.g., [27, 36].

1.3 New results

The main contribution of this paper is the first thorough study of the online Facility Location problem with deletions and design of new algorithms for this model in several natural settings.

We begin with the study of the simplest, *uncapacitated* model. We present in Theorem 2.2 an online $O(\log \mathfrak{n}_{\mathfrak{act}}/\log\log \mathfrak{n}_{\mathfrak{act}})$-competitive algorithm for the uncapacitated Facility Location problem with deletions, where $\mathfrak{n}_{\mathfrak{act}}$ is the number of active clients at the end of the input sequence; this bound gives an asymptotically optimal competitive ratio. Our algorithm is an extension of the classical insertion-only algorithm for the Facility Location problem due to Meyerson [40], and we show that one can modify the analysis from [40] to allow deletions.

Next, we turn our attention to the *capacitated* Facility Location problem. We first prove (Observation 3.1) that if *no deletions* are permitted, then a simple modification of Meyerson's algorithm for online Facility Location can be applied to achieve an optimal competitive ratio of $O(\log \mathfrak{n}/\log\log \mathfrak{n})$. However, when deletions are allowed, then the capacitated version of the problem is significantly more challenging than the uncapacitated one. We show that still, using more involved approach incorporating hierarchically well-separated trees, one can obtain an online $O(\log \mathfrak{m} + \log \mathfrak{c} \log \mathfrak{n})$-competitive algorithm for the capacitated Facility Location problem with deletions, in the fully dynamic model, where $\mathfrak{n}$ is the number of queries, $\mathfrak{m}$ is the number of points in the input metric, and $\mathfrak{c}$ is the capacity of the facilities (Theorem 4.4).

We notice that while the algorithms from Theorem 2.2 and Observation 3.1 do not need to know the input metric, the result from Theorem 4.4 assumes that the input metric is known to the algorithm.

Our work demonstrates that despite the fact that the online Facility Location problem with deletion is clearly more complex than the classical online problem with insertions only, the most natural variants of these problems, for both the uncapacitated and the capacitated model for uniform facility costs, have very efficient online algorithms that achieve competitive ratios matching or almost matching those of the insertion only variants of the problem.

2 Online uncapacitated facility location

We begin with the study of the simplest, *uncapacitated* model, and describe an insertion-only online algorithm for uncapacitated Facility Location due to Meyerson [40].

Algorithm M:

When a new demand point x arrives then find its nearest open facility y and set $d_x = \min\{\mathsf{dist}(x, y), 1\}$. Next, with probability d_x open a new facility at point x and assign x to it; otherwise, assign x to y.

Meyerson [40] proved that Algorithm M is $O(\log \mathfrak{n}/ \log\log \mathfrak{n})$-competitive in the model with insertions only (see also [22, 23]).

The most natural approach to obtain an online algorithm for the Facility Location problem with deletions is to attempt to modify Algorithm M. Indeed, there is a simple modification of Algorithm M that addresses the deletions. The following Algorithm M* proceeds as in Algorithm M, except that when a facility is removed, it reprocesses all the points that are now not assigned to any facility using the original algorithm.

Algorithm M*:

When a new demand point x arrives then find its nearest open facility y and set $d_x = \min\{\mathsf{dist}(x, y), 1\}$. Next, with probability d_x open a new facility at point x and assign x to it; otherwise, assign x to y.

When a point x that is an open facility is to be deleted, then reassign all points assigned to x using the algorithm for the insertions.

While it is appealing to hope that Algorithm M* has performance similar to that of Algorithm M for insertions only, but in fact asymptotically, it does no better than opening facilities at all points.

▶ **Claim 2.1.** *Algorithm M* has competitive ratio of* $\Omega(\mathfrak{n}_{\mathfrak{act}})$.

Proof. For any $k \in \mathbb{N}_+$, consider a star metric with center $\mathfrak{o}$ connected to points $\mathcal{X} = \{x_1, \dots, x_k\}$ at distance $\varepsilon = \frac{1}{k}$ from $\mathfrak{o}$. Consider the following input sequence:

1. Add k^2 clients $a_1, \dots, a_{k^2}$ located at $\mathfrak{o}$.
2. For each $i = 1, \dots, k$ add a client b_i located at x_i.
3. Remove the clients $a_1, \dots, a_{k^2-1}$ in that order.

We will analyze the performance of algorithm M^* in the above scenario. While doing that, we assume that whenever clients are reassigned, they are reassigned in the order in which they were originally assigned.

Consider the first two stages, when all the clients are added. In step 1, client a_1 opens a facility and then all other clients at $\mathfrak{o}$ connect to a_1. In step 2, each b_i opens a facility with probability ε and connects to a_1 otherwise. For the remainder of the analysis, let us assume that at least one of the clients b_j opens a facility. This happens with probability $p_1 = 1 - (1 - \varepsilon)^k = \Omega(1)$.

Next, we remove a_1 and reassign all other clients assigned to a_1. One by one, each client at $\mathfrak{o}$ flips a coin and with probability ε connects to one of the b_i's; otherwise it opens a facility. As soon as some a_i opens a facility, all other clients at $\mathfrak{o}$ connect to a_i, and each client at $\mathcal{X}$ that has not yet opened a facility (and hence was initially connected to a_1) does so with probability ε; otherwise it connects to a_i. If no a_i opens a facility, then each client at $\mathcal{X}$ that has not yet opened a facility, opens one with probability 2ε, and otherwise it connects to one already open facility at some b_j.

We then remove all other clients $a_2, \dots, a_{k^2-1}$. Each time the client removed has a facility opened, the scenario described in the previous paragraph repeats. Removal of a client with no facility opened has, of course, no effect on the other clients.

Note that each time we remove a client at $\mathfrak{o}$ that has a facility open, each of the b_j's opens a facility with probability at least ε, unless it has already opened a facility. How many times does that happen? Let us count the number of a_i's with $2 \leq i \leq k^2 - 1$ that at some moment open a facility (and then get removed). Each of the a_i's flips a coin exactly once (which corresponds to the situation that at some moment we have already removed $a_1, \ldots, a_j$ for some $j < i$, and there is no facility open at $a_{j+1}, \ldots, a_{i-1}$, with points $a_{j+1}, \ldots, a_{i-1}$ connected to facilities from $\mathcal{X}$). Furthermore, the coin flip for a_i is independent from coin flips made by $a_2, \ldots, a_{i-1}, a_{i+1}, \ldots, a_{k^2-1}$. Therefore the number of a_i's that at some moment open a facility and then get removed has the binomial distribution with parameters $k^2 - 2$ and ε. Hence, by Chernoff bound, with probability $p_2 \geq 1 - e^{-\varepsilon(k^2-2)/6} = \Omega(1)$, there are at least $\varepsilon(k^2-2)/2 \geq k/2 - 1$ of those clients. Therefore, we can conclude that each b_j opens a facility with probability at least $p_3 = 1 - (1-\varepsilon)^{k/2-1} = \Omega(1)$.

The optimal offline solution opens a facility at a_{k^2} and connects all points $b_1, \ldots, b_k$ to a_{k^2}, and it has cost $1 + \varepsilon k = 2$. On the other hand, with probability $p_1 p_2 = \Omega(1)$ algorithm M^* opens at least $kp_3 = \Omega(k)$ facilities from $b_1, \ldots, b_k$ in expectation. This gives an $\Omega(k) = \Omega(\mathfrak{n}_{\mathfrak{act}})$ competitive ratio. So asymptotically M^* does no better than just opening facilities at all points. ◀

2.1 Asymptotically optimal competitive ratio for uncapacitated facility location with deletions

Claim 2.1 shows one of the main challenges that online algorithms with deletions must cope with: we cannot let points attempt to open a facility too frequently, since then we would open too many facilities, and at the same time we have to be able to open some facilities in the neighborhood of the facilities that we are closing. To address this challenge we have to provide a delicate online strategy that will maintain a right balance between these two desirables.

Algorithm 1:

Newly arriving points: When a new demand point x arrives then find its nearest open facility y and set $d' = \min\{\mathsf{dist}(x,y), 1\}$. With probability d' open a new facility at point x and assign x to it; otherwise, assign x to y and set $p_x := d'$. (The algorithm will memorize p_x for future use.)

Deletion: When a point that is *not an open facility* is to be deleted, then just remove that point.

When a point that is an *open facility* is to be deleted, then reassign all points assigned to it: When a point x is to be reassigned, then find its nearest open facility y and set $d' = \min\{\mathsf{dist}(x,y), 1\}$. Let p_x be the last value used in the processing of x. If $d' \leq 2p_x$, then assign x to y; otherwise with probability d' open a new facility at x; else, with probability $1 - d'$ assign x to y and set $p_x := d'$.

The following theorem analyzes the performance of Algorithm 1.

▶ **Theorem 2.2.** *Algorithm 1 is $O(\log \mathfrak{n}_{\mathfrak{act}} / \log\log \mathfrak{n}_{\mathfrak{act}})$-competitive, where $\mathfrak{n}_{\mathfrak{act}}$ is the number of active clients at the end of the input sequence (in particular, as $\mathfrak{n}$ is the input length, we have $\mathfrak{n}_{\mathfrak{act}} \leq \mathfrak{n}$).*

Proof. We follow the approach of Meyerson [40], with special care taken to deal with deletions.

Consider any optimal solution. For a fixed facility v, let S be the set of clients connected to v in the optimal solution. Let d^* be the average distance from the points of S to v. We split S into $h+1$ subsets $S_0, S_1, \ldots, S_h$, where $h = \lceil 2\log|S|/\log\log|S|\rceil$. Points in S_0 are at distance at most d^* from v. For $1 \le i \le h$, points in S_i are at distance greater than $d^*(\log|S|)^{(i-1)/2}$, but at most $d^*(\log|S|)^{i/2}$ from v. Note that each point is contained in some S_i, as $d^*(\log|S|)^{h/2} \ge d^*|S|$, and a single client at distance more than $d^*|S|$ would contradict the average distance d^*.

For each set S_i we split the time into two epochs: the second epoch starts when the first point in S_i becomes an open facility in the solution of the algorithm – note that the second epoch might never start.

Consider the first epoch of some S_i. The part of the cost of the final solution incurred by the points in S_i during the first epoch is the total connection cost of these points at the end of the first epoch plus possibly a single opening cost. The connection cost of a point $x \in S_i$ is upper bounded by twice the probability of the last coin flip of x, regardless of whether that connection was preceded by a coin flip or not. To bound the total connection cost of S_i in the first epoch it is therefore enough to bound the sum of the probabilities of all the coin flips related to S_i made during that epoch.

▶ **Lemma 2.3.** *The expected value of the sum of the probabilities of the coin flips related to points in S_i and made before the first facility in S_i opens is at most* 1.

Proof. Let $P_1, P_2, \ldots, P_M$ be the probabilities of all coin flips (with non-zero probabilities) made for points in S_i and let $X_1, X_2, \ldots, X_M$ be the outcomes of these coin flips, so that $P[X_j = 1] = P_j$ (note that each P_j is a random variable, and not a constant since its value might possibly depend on earlier coin flips). Also, add to both sequences a virtual „sentinel" coin flip with $P_{M+1} = 1$, $X_{M+1} = 1$. Define $Z_0 = 0$ and $Z_{j+1} = Z_j + P_{j+1} - X_{j+1}$ for $j = 1, \ldots, M+1$. Then the sequence $\{Z_j\}$ forms a martingale. Let $T = \min\{j > 0 : X_j = 1\}$ be the position of the first heads in $\{X_j\}$. Note, that T is well defined, since $X_{M+1} = 1$. Also note, that $T = \min\{j > 0 : Z_j \le Z_{j-1}\}$, i.e., T is a stopping time for $\{Z_j\}$. Since T is bounded, from the Doob's optional stopping time theorem we get that $E[Z_T] = E[Z_0] = 0$. However, we also have

$$Z_T = \sum_{j=1}^{T} P_j - 1,$$

and thus the claim follows. Note that the claim does not hold with equality, since the above expression might include the virtual coin flip added to make T well defined. ◀

It follows that the total cost incurred by the points of S_i during the first epoch is a constant, so the overall cost of the first epoch is $O(h)$.

Consider now the cost incurred by any point $x \in S_i$ in the second epoch. If x does not make any coin flips in the second epoch, then any connection made by x has cost upper bounded by $2d^*(\log|S|)^{i/2}$, since there is an open facility in S_i at this point. Consider now the case when x does make a coin flip during the second epoch. We then upper bound the cost of the last connection made by x by twice the probability of the last coin flip of x. We also need to bound the cost of a facility that x might potentially open. The expected number of facilities opened by x is the sum of the probabilities of all its coin flips. Each term in this sum is at least twice the previous one, and so the expected number of facilities opened by x is at most twice the probability of the last coin flip of x. This probability is at most $2d^*(\log|S|)^{i/2}$ since there is an open facility in S_i.

To sum up, the total cost incurred by $x \in S_i$ during the second epoch is at most $8d^*(\log|S|)^{i/2}$. Consider first the case where S_i is not the innermost layer, i.e., $0 < i \leq h$. Any $x \in S_i$ is then connected to v in the optimum solution, and thus incurs a cost of at least $d^*(\log|S|)^{(i-1)/2}$. Therefore, the cost incurred by such x in the second epoch is at most $\sqrt{\log|S|}$ times the corresponding cost in the optimal solution. Consider now the innermost layer. For any client in S_0 Algorithm 1 pays at most $O(d^*)$, leading to $O(d^*|S_0|)$ total cost of the second epoch, which by the definition of d^* is at most a constant factor more than the connection cost payed by the optimal algorithm.

Note that in the analysis we completely ignore the cost generated by the clients, that are removed during the course of the algorithm, as those clients in the end generate no cost to Algorithm 1. ◀

Since even in the incremental model (when points can only arrive, not vanish) there is an $\Omega(\log \mathfrak{n}/\log\log \mathfrak{n})$-lower bound (cf. [22]), Algorithm 1 is optimal up to a constant factor.

3 Capacitated online facility location (with insertions only)

Our result in Theorem 2.2 shows that for uncapacitated Facility Location with deletions, one can extend the approach from earlier works (see [40] and also [22]) to design online algorithms that achieve asymptotically optimal competitive ratio. However, the model of *capacitated* Facility Location with deletions is significantly more complex. Still, in the most basic case, the model *with insertions only*, we observe that Algorithm M can be extended to the model of capacitated Facility Location. We run Algorithm M with a single modification: the nearest facility y now is the nearest facility that is not fully saturated, that is, that has still available capacity.

▶ **Observation 3.1.** *Algorithm M in the capacitated case is $O(\log \mathfrak{n}/\log\log \mathfrak{n})$-competitive in the model with insertions only.*

Proof. We only sketch the proof, since it is a straightforward modification of the original proof of Meyerson [40] and Fotakis [22]. Similarly as described in the proof of Theorem 2.2, we partition the set S of clients connected to some open facility v into subsets $S_0, \ldots, S_h$, depending on their distance to v. What is different from the analysis of Meyerson and Fotakis is the way in which we split time into epochs for each layer S_i, as here the first and second epoch can possibly interlace. Formally, the cost incurred by a client belongs to the first epoch, if at the moment when the client appears there is no unsaturated open facility in the layer S_i. If there is an unsaturated open facility in the layer S_i, then the cost incurred by the client is classified to the second epoch.

The total cost of clients of the first epoch is bounded by Lemma 2.3, and is at most $\ell + \mathfrak{n}/\mathfrak{c}$, where ℓ is the total number of sets S_i, as at most $\mathfrak{n}/\mathfrak{c}$ facilities may be saturated during the course of the algorithm. Note that ℓ is exactly $(h+1) = O(\log \mathfrak{n}/\log\log \mathfrak{n})$ times greater than the number of open facilities in the optimal solution. Also, as each facility can serve only $\mathfrak{c}$ clients, the term $\mathfrak{n}/\mathfrak{c}$ is not greater than the cost of the optimal solution.

The cost of clients of the second epoch is bounded as in the proof of Theorem 2.2, i.e., Algorithm M pays at most $d^*(\log|S|)^{i/2}$ for each client from S_i, whereas optimal solution pays at least $d^*(\log|S|)^{(i-1)/2}$, assuming $i > 0$. The cost of clients from S_0 is bounded analogously as in the proof of Theorem 2.2. ◀

4 Capacitated facility location with deletions

The result in Section 3 may give a hope that also our result from Theorem 2.2 for uncapacitated Facility Location with deletions can be easily extended to the model of *capacitated* Facility Location with deletions. For example, let us think of a simple extension of Algorithm 1 to the capacitated case. Perhaps the most natural idea is to let d be the distance to the closest open unsaturated facility, i.e., a facility which still can serve additional clients. Unfortunately this line of reasoning does not lead to a meaningful competitive ratio, because in the case when all the clients arrive at the same location all the distances are equal to zero and therefore such a modified version of Algorithm 1 would be deterministic. Playing against a deterministic algorithm is very convenient for the adversary, as the adversary might remove all the clients which were not turned into open facilities by the algorithm, leading to $\Omega(\mathfrak{c})$ competitive ratio.

One natural idea to introduce randomness to the algorithm is to increase the value of d, so that even if the distance to the closest facility is zero, the client might still decide to open a new facility. However, this idea alone does not seem to lead to any reasonable competitive factor. Below, we present a typical hard example for one possible implementation of this idea. (Note, that this competitive ratio is as bad as the one obtained by an algorithm that opens facilities in all input points.)

▶ **Claim 4.1.** *Let $\mathcal{A}$ be Algorithm 1 modified, so that d is the maximum of the distance to the closest unsaturated facility and $10/\mathfrak{c}$. Then, the competitive ratio of $\mathcal{A}$ is $\Omega(\mathfrak{c})$.*

Proof. Consider a star metric with the center $\mathfrak{o}$ and the remaining $10\mathfrak{c}^2$ points at distance $\frac{1}{2}$ from $\mathfrak{o}$. Let us analyze the performance of $\mathcal{A}$ on the following input sequence. First, we repeat $\mathfrak{c}$ times the following insertion: insert a point at $\mathfrak{o}$ and then $10\mathfrak{c}$ points in different leaves of the metric, and then, at the end, remove all points located in the leaves.

The optimal solution will have one open facility at one of the $\mathfrak{c}$ points located at $\mathfrak{o}$, and so the optimal cost is 1. We claim that the expected size of the solution found by $\mathcal{A}$ is $\Omega(\mathfrak{c})$.

To see this, consider a single round of insertions. If at the beginning of the round, there are no unsaturated facilities at $\mathfrak{o}$, then one is created with probability $\Theta(1)$. If that happens, then each of the clients inserted at the leaves opens a facility with probability $\frac{1}{2} + 10/\mathfrak{c}$ until the facility is saturated. Since there are $10\mathfrak{c}$ such clients, w.h.p. the facility gets saturated. It also follows, that for any round, w.h.p. there are no unsaturated facilities at $\mathfrak{o}$ at the beginning of the round.

The expected number of facilities opened at $\mathfrak{o}$ is the sum over all rounds of insertions of the probabilities that a facility is open at the beginning of the round. Based on the observations of the previous paragraph, this probability is $\Theta(1)$ for each round, and the claim follows. ◀

4.1 Hierarchically well-separated trees and facility location

We will present an online algorithm for the *capacitated* Facility Location problem in the fully dynamic setting with low competitive ratio, with both insertions and deletions. We begin with a brief overview. We will assume that (unlike in the rest of the paper) *the input metric is given in advance*, $\mathcal{V}$ is the set of all points in the metric space, and $\mathfrak{m}$ is the number of points in the metric space, $\mathfrak{m} = |\mathcal{V}|$. We use the embedding of the original metric space into hierarchically well-separated trees (cf. [9, 10, 18]), on which we run our Facility Location algorithm. Once we run the algorithm on a hierarchically well-separated tree $\mathfrak{T}$, for every open facility, we will evenly split its capacity into $\mathfrak{h} = O(\log \mathfrak{c})$ parts and then allocate each partial capacity solely to the points in one of $\mathfrak{h}$ areas we will define later. Then we use a key property which ensures that every point v that would use an open facility u in the offline uncapacitated optimal solution, to find a replacement facility with still available capacity that is at the same distance from v in $\mathfrak{T}$ as the distance from u to v in $\mathfrak{T}$.

Our approach relies on the concept of *hierarchically well-separated trees (HSTs)*, which are metric spaces defined on the leaves of weighted rooted trees (cf. [9, 10, 18]). It is known (see [18]) that for every metric space, there is an HST with stretch $O(\log \mathfrak{m})$. Let $\mathfrak{T}$ be such an HST for the metric given in our instance. Let the level of an internal node in the tree $\mathfrak{T}$ be the number of edges on the path to the root. Let Δ denote the diameter of the resulting metric space. In our paper, we will assume, without loss of generality, that the diameter of the original metric space is $\Delta = 1$. (Indeed, if a pair of points is at distance larger than 1, then in the Facility Location problem we will never allocate one of them to another, since it is always cheaper to pay 1 to open a new facility; therefore, we can treat any distance larger than 1 as equal to 1.) We also modify short distances in the metric, that is, for any pair of points in the metric we assume their distance is at least $1/\mathfrak{c}$. Note that as $\mathfrak{n}_{\mathfrak{act}}/\mathfrak{c}$ is a lower bound on the cost of the optimum solution such modification of the metric increases the cost of the optimum solution at most by a constant factor. Then using the framework of HSTs, we will assume that the metric in the instance of capacitated online Facility Location we are solving is a shortest paths metric in a tree $\mathfrak{T}$ of depth $\mathfrak{h} = \log \mathfrak{c}$, satisfying the following conditions:

- any edge connecting vertices of depth i and $i+1$ is of length 2^{-i},
- the set of potential clients are leaves of $\mathfrak{T}$,
- all leaves of $\mathfrak{T}$ are at the same depth $\mathfrak{h}$.

▶ **Definition 4.2.** For two leaves u, v of $\mathfrak{T}$, define $\mathsf{dist\_log}(u, v) = \lfloor -\log \mathsf{dist}_{\mathfrak{T}}(u, v)\rfloor$.

Less formally, $\mathsf{dist\_log}(u, v)$ is the depth of the lowest common ancestor of u and v in $\mathfrak{T}$, which follows from the assumption that the weights of edges of $\mathfrak{T}$ are powers of two depending on their depth. From the triangle inequality we have the following property.

▶ **Claim 4.3.** *For $u, v, w \in \mathcal{V}$ we have* $\mathsf{dist\_log}(u, w) \geq \min\{\mathsf{dist\_log}(u, v), \mathsf{dist\_log}(v, w)\}$.

4.2 Algorithm for fully dynamic capacitated facility location in HSTs

In this section, we present an algorithm for fully dynamic capacitated Facility Location in a hierarchically well-separated tree $\mathfrak{T}$, Algorithm 2. We will analyze its performance in the next section.

We will assume the input metric space is the shortest path metric in a hierarchically well-separated tree $\mathfrak{T}$, with all input points coming from the leaves of $\mathfrak{T}$, as defined in the previous section. In the algorithm below, when a new facility is opened at a point v, then its capacity $\mathfrak{c}$ is evenly split into $\mathfrak{h}$ parts, denoted as functions $cap_i(v)$ for $0 \leq i < \mathfrak{h}$. We will design the algorithm so that the capacity $cap_i(v)$ will be used solely by clients u such that $\mathsf{dist\_log}(u, v) = i$, that is, by the clients such that the lowest common ancestor of u and v in $\mathfrak{T}$ is of depth i. Apart from that constraint, the algorithm mimics Algorithm 1 from Section 2.1.

Algorithm 2:

```
insert $v$:
  - Call connect($v$).
delete $v$:
  - For all clients of $v$ (in arbitrary order) call connect($u$).
```

connect v:

- If $\max_v$ is undefined, then set $\max_v = 0$.
- Let u be the closest point to v such that $cap_i(u) > 0$, where $i = \mathsf{dist\_log}(u, v)$.
- If such u does not exist, then **open**(v) and **exit**.
- $p = \min\{1, \mathsf{dist}_{\mathfrak{T}}(u, v) + \frac{12\ \mathfrak{h} \ln \mathfrak{n}}{\mathfrak{c}}\}$.
- If $p \leq 2 \max_v$, then connect v to u and decrease $cap_i(u)$ by one.
- Otherwise: (i) set $\max_v = p$, (ii) with probability p **open**(v), and with probability $1 - p$ connect v to u and decrease $cap_i(u)$ by one.

open v:

- Open a facility at v.
- For each $0 \leq i < \mathfrak{h}$ set $cap_i(v) = \lfloor \mathfrak{c}/\mathfrak{h} \rfloor$.

In the full version of this paper we prove the following main result for the capacitated case of Facility Location with deletion.

▶ **Theorem 4.4.** *There is an $O(\log \mathfrak{m} + \log \mathfrak{c} \log \mathfrak{n})$-competitive online algorithm for capacitated* Facility Location *with deletions.*

5 Conclusions

In this paper we present the first thorough study of natural variants of the online Facility Location problem, when the clients and facilities are not only allowed to arrive to the system, but they can also depart from the system at any moment. In this fully-dynamic online problem, we study two fundamental settings: uncapacitated and capacitated Facility Location for uniform facility costs. For uncapacitated Facility Location, we design an extension of the classical insertion-only randomized algorithm for the Facility Location problem due to Meyerson [40], and show that it achieves an asymptotically optimal competitive ratio of $O(\log \mathfrak{n}_{\mathfrak{act}} / \log \log \mathfrak{n}_{\mathfrak{act}})$ (Theorem 2.2). The capacitated Facility Location is more complex, and here we first show (Observation 3.1) that if no deletions are allowed, then one can achieve an asymptotically optimal competitive ratio of $O(\log \mathfrak{n} / \log \log \mathfrak{n})$, the same bound as it is known for the uncapacitated variant. When deletions are allowed, the task is more challenging, but we still are able to incorporate the framework of hierarchically well-separated trees to obtain an online $O(\log \mathfrak{m} + \log \mathfrak{c} \log \mathfrak{n})$-competitive algorithm for the capacitated Facility Location problem with deletions (Theorem 4.4).

Our work demonstrates that despite the fact that the online Facility Location problem with deletion is clearly more complex than the classical online problem with insertions only, the most natural variants of these problems, for both the uncapacitated and the capacitated model for uniform facility costs, have very efficient online algorithms that achieve competitive ratios matching or almost matching those of the insertion only variants of the problem. It is an interesting open problem whether one can improve the competitive ratio for the capacitated case with deletions, in particular whether it is possible to remove the dependence on $\mathfrak{m}$.

References

1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2012)*, pages 459–467. Society for Industrial and Applied Mathematics, 2012.

2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Spectral sparsification in dynamic graph streams. In *Proceedings of the 17th International Workshop on Approximation, Randomization, and Combinatorial Optimization (RANDOM'2013)*, pages 1–10. Springer Verlag, Berlin, Heidelberg, 2013.

3 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph (Seffi) Naor. A general approach to online network optimization problems. *ACM Transactions on Algorithms*, 2(4):640–660, 2006. doi:10.1145/1198513.1198522.

4 Hyung-Chan An, Ashkan Norouzi-Fard, and Ola Svensson. Dynamic facility location via exponential clocks. *ACM Transactions on Algorithms*, 13(2):21:1–21:20, 2017. doi:10.1145/2928272.

5 Hyung-Chan An, Mohit Singh, and Ola Svensson. LP-based algorithms for capacitated facility location. *SIAM Journal on Computing*, 46(1):272–306, 2017. doi:10.1137/151002320.

6 Aris Anagnostopoulos, Russell Bent, Eli Upfal, and Pascal Van Hentenryck. A simple and deterministic competitive algorithm for online facility location. *Information and Computation*, 194(2):175–202, 2004. doi:10.1016/j.ic.2004.06.002.

7 Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Piotr Sankowski. Online network design with outliers. *Algorithmica*, 76(1):88–109, 2016. doi:10.1007/s00453-015-0021-y.

8 Mihai Badoiu, Artur Czumaj, Piotr Indyk, and Christian Sohler. Facility location in sublinear time. In *Proceedings of the 32nd Annual International Colloquium on Automata, Languages and Programming (ICALP'2005)*, volume 3580 of *Lecture Notes in Computer Science*, pages 866–877. Springer Verlag, Berlin, Heidelberg, 2005. doi:10.1007/11523468_70.

9 Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science, (FOCS'1996)*, pages 184–193. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548477.

10 Yair Bartal. On approximating arbitrary metrices by tree metrics. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC'1998)*, pages 161–168. ACM Press, 1998. doi:10.1145/276698.276725.

11 Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

12 Jaroslaw Byrka and Karen Aardal. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. *SIAM Journal on Computing*, 39(6):2212–2231, 2010. doi:10.1137/070708901.

13 Graham Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In *Proceedings of the 24th ACM Symposium on Principles of Database Systems (PODS'2005)*, pages 271–282. ACM, 2005. doi:10.1145/1065167.1065201.

14 Gérard Cornuéjols, George L. Nemhauser, and Lairemce A. Wolsey. The uncapacitated facility location problem. In Pitu B. Mirchandani and Richard L. Francis, editors, *Discrete Location Theory*, pages 119–171. John Wiley and Son, Inc., New York, 1990.

15 Artur Czumaj, Christiane Lammersen, Morteza Monemizadeh, and Christian Sohler. $(1+\varepsilon)$-approximation for facility location in data streams. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2013)*, pages 1710–1728. Society for Industrial and Applied Mathematics, 2013. doi:10.1137/1.9781611973105.123.

16 Wenqiang Dai and Xianju Zeng. Incremental facility location problem and its competitive algorithms. *Journal of Combinatorial Optimization*, 20(3):307–320, 2010. doi:10.1007/s10878-009-9219-8.

17 Gabriella Divéki and Csanád Imreh. Online facility location with facility movements. *Central European Journal of Operations Research*, 19(2):191–200, June 2011. doi:10.1007/s10100-010-0153-8.

18 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, November 2004. doi:10.1016/j.jcss.2004.04.011.

19 Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

20 Dimitris Fotakis. Incremental algorithms for facility location and k-median. *Theoretical Computer Science*, 361(2-3):275–313, 2006. doi:10.1016/j.tcs.2006.05.015.

21 Dimitris Fotakis. A primal-dual algorithm for online non-uniform facility location. *Journal of Discrete Algorithms*, 5(1):141–148, 2007. doi:10.1016/j.jda.2006.03.001.

22 Dimitris Fotakis. On the competitive ratio for online facility location. *Algorithmica*, 50(1):1–57, 2008. doi:10.1007/s00453-007-9049-y.

23 Dimitris Fotakis. Online and incremental algorithms for facility location. *SIGACT News*, 42(1):97–131, 2011.

24 Gereon Frahling and Christian Sohler. Coresets in dynamic geometric data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC'2005)*, pages 209–217. ACM Press, 2005. doi:10.1145/1060590.1060622.

25 Naveen Garg, Anupam Gupta, Stefano Leonardi, and Piotr Sankowski. Stochastic analyses for online combinatorial optimization problems. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2008)*, pages 942–951. Society for Industrial and Applied Mathematics, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347185.

26 Fabrizio Grandoni, Anupam Gupta, Stefano Leonardi, Pauli Miettinen, Piotr Sankowski, and Mohit Singh. Set covering with our eyes closed. *SIAM Journal on Computing*, 42(3):808–830, 2013. doi:10.1137/100802888.

27 Anupam Gupta and Amit Kumar. Online Steiner tree with deletions. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA'2014)*, pages 455–467. Society for Industrial and Applied Mathematics, 2014. doi:10.1137/1.9781611973402.34.

28 Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC'2004)*, pages 373–380. ACM Press, 2004. doi:10.1145/1007352.1007413.

29 Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

30 Gene Kan. Chapter 8: Gnutella. In Andy Oram, editor, *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*. O'Reilly & Associates, 2001.

31 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *SIAM Journal on Computing*, 46(1):456–477, 2017. doi:10.1137/141002281.

32 Michael Kapralov and David P. Woodruff. Spanners and sparsifiers in dynamic streams. In *Proceedings of the 33rd ACM Symposium on Principles of Distributed Computing (PODC'2014)*, pages 272–281. ACM Press, 2014.

33 Peter Kling, Friedhelm Meyer auf der Heide, and Peter Pietrzyk. An algorithm for online facility leasing. In *Proceedings of the 19th International Colloquium on Structural Information and Communication Complexity (SIROCCO'2012)*, pages 61–72. Springer Verlag, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-31104-8_6.

34 John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishan Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. OceanStore: An architecture for global-scale persistent storage. *SIGPLAN Notices*, 35(11):190–201, 2000. doi:10.1145/356989.357007.

35 Christiane Lammersen and Christian Sohler. Facility location in dynamic geometric data streams. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA'2008)*, pages 660–671. Springer Verlag, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-87744-8_55.

36 Jakub Łącki, Jakub Oćwieja, Marcin Pilipczuk, Piotr Sankowski, and Anna Zych. The power of dynamic distance oracles: Efficient dynamic algorithms for the Steiner tree. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC'2015)*, pages 11–20. ACM Press, 2015. doi:10.1145/2746539.2746615.

37 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45–58, 2013. doi:10.1016/j.ic.2012.01.007.

38 Andrew McGregor. Graph stream algorithms: A survey. *SIGMOD Record*, 43(1):9–20, 2014. doi:10.1145/2627692.2627694.

39 Ramgopal R. Mettu and C. Greg Plaxton. The online median problem. *SIAM Journal on Computing*, 32(3):816–832, 2003. doi:10.1137/S0097539701383443.

40 Adam Meyerson. Online facility location. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS'2001)*, pages 426–431. IEEE Computer Society, 2001. URL: http://dl.acm.org/citation.cfm?id=874063.875567.

41 Chandrashekhar Nagarajan and David P. Williamson. Offline and online facility leasing. *Discrete Optimization*, 10(4):361–370, 2013. doi:10.1016/j.disopt.2013.10.001.

42 Daniel J. Rosenkrantz, Giri K. Tayi, and S.S. Ravi. Obtaining online approximation algorithms for facility dispersion from offline algorithms. *Networks*, 47(4):206–217, 2006. doi:10.1002/net.20109.

43 Mário César San Felice, David P. Williamson, and Orlando Lee. A randomized $O(\log n)$-competitive algorithm for the online connected facility location problem. *Algorithmica*, 76(4):1139–1157, 2016. doi:10.1007/s00453-016-0115-1.

44 Rajesh Sharma and Anwitaman Datta. Supernova: Super-peers based architecture for decentralized online social networks. In *Proceedings of the 4th Fourth International Conference on Communication Systems and Networks, (COMSNETS'2012)*, pages 1–10. IEEE, 2012. doi:10.1109/COMSNETS.2012.6151349.

45 David B. Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC'1997)*, pages 265–274. ACM Press, 1997. doi:10.1145/258533.258600.

46 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985. doi:10.1145/2786.2793.

47 Jouni Smed, Timo Kaukoranta, and Harri Hakonen. Networking and multiplayer computer games - the story so far. *International Journal of Intelligent Games & Simulation*, 2(2):101–110, 2003.

48 Seeun Umboh. Online network design algorithms via hierarchical decompositions. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2015)*, pages 1373–1387. Society for Industrial and Applied Mathematics, 2015. doi:10.1137/1.9781611973730.91.

49 Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering (ICDE'2003)*, pages 49–60. IEEE Computer Society, 2003.

Improved Routing on the Delaunay Triangulation

Nicolas Bonichon
Université de Bordeaux, LaBRI, UMR 5800, F-33400 Talence, France

Prosenjit Bose
Carleton University, 1125 Colonel By Dr, Ottawa, ON, Canada

Jean-Lou De Carufel
University of Ottawa, 800 King Edward Ave, Ottawa, ON, Canada

Vincent Despré
Team Gamble, INRIA Nancy, 54600 Villers-lès-Nancy, France

Darryl Hill
Carleton University, 1125 Colonel By Dr, Ottawa, ON, Canada

Michiel Smid
Carleton University, 1125 Colonel By Dr, Ottawa, ON, Canada

Abstract

A geometric graph $G = (P, E)$ is a set of points in the plane and edges between pairs of points, where the weight of an edge is equal to the Euclidean distance between its two endpoints. In local routing we find a path through G from a source vertex s to a destination vertex t, using only knowledge of the current vertex, its incident edges, and the locations of s and t. We present an algorithm for local routing on the Delaunay triangulation, and show that it finds a path between a source vertex s and a target vertex t that is not longer than $3.56|st|$, improving the previous bound of $5.9|st|$.

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

Keywords and phrases Delaunay, local routing, geometric, graph

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.22

Funding This work was supported by the National Sciences and Engineering Research Council of Canada (NSERC) and by the French ANR grant ASPAG (ANR-17-CE40-0017)

1 Introduction

A Euclidean geometric graph $G = (P, E)$ is a set P of points embedded in the plane, and a set E of edges, where each $e \in E$ is a pair of points (u, v) in P, and the weight of e is the Euclidean distance $|uv|$.

A *local routing algorithm* A is an algorithm that routes a packet through the geometric graph G from a source vertex s to a target vertex t using only knowledge of the locations of s and t, as well as the location of the current vertex and its adjacent vertices. Let $\mathcal{P}\langle s, t\rangle$ be the path found in G from s to t using A. The *routing ratio* of A for any two points s and t in the geometric graph G is the ratio of the length of $\mathcal{P}\langle s, t\rangle$ to the Euclidean distance from s to t. An algorithm A has a routing ratio c for a class of geometric graphs $\mathcal{G}$, if, for any two vertices s and t in $G \in \mathcal{G}$, $|\mathcal{P}\langle s, t\rangle| \leq c \cdot |st|$.

A graph $G = (P, E)$ is a c-spanner if for any pair of points u and v in P the shortest path in G is not longer than $c|uv|$. The value c is referred to as the *stretch factor* or *spanning*

© Nicolas Bonichon, Prosenjit Bose, Jean-Lou De Carufel, Vincent Despré, Darryl Hill, and Michiel Smid;
licensed under Creative Commons License CC-BY

Table 1 Spanning and Routing Ratios of Delaunay Triangulations. Tight results are shown in bold.

Graph	Spanning Ratio	Routing Ratio
TD-Delaunay	**2** [8]	$\mathbf{5/\sqrt{3} \approx 2.89}$ [5]
L_1 and L_∞-Delaunay	$\mathbf{\sqrt{4+2\sqrt{2}} \approx 2.61}$ [3]	$\sqrt{10} \approx 3.16$ [7]
$Hexagon$-Delaunay	**2** [9]	
L_2-Delaunay	1.998 [13]	3.56 (this paper)

ratio of G. The stretch factor of G is thus a lower bound on the routing ratio of G for any routing algorithm A, and the routing ratio is an upper bound on the spanning ratio of G. Geometric spanners are described in detail in the book by Narasimhan and Smid [12].

A notable geometric graph is the *Delaunay triangulation*. Given a set P of points in the plane, we construct the Delaunay triangulation of P as follows. For each triple (p, q, r) of points in P, let C be the circle through p, q, and r. If there are no points of P in the interior of C, then we connect p, q, and r by edges to form a triangle. In this paper we assume that P is in general position: no 3 points are colinar and no 4 points are cocircular.

The Delaunay triangulation was first proven to be a spanner by Dobkin et al. [10], who showed an upper bound of 5.08 on the spanning ratio. This was subsequently improved to 2.42 by Keil and Gutwin [11], and then to 1.998 by Xia [13]. Xia and Zhang proved later that there exist Delaunay triangulations with spanning ratio greater than 1.59 [14].

Bose and Morin [6] explored some of the theoretical limitations of routing, and provided some of the first deterministic routing algorithms with constant routing ratio on the Delaunay triangulation. They denoted the spanning ratio found by Dobkin et al. [10] as $c_{dfs} \approx 5.08$. They showed that it is possible to locally route on the Delaunay triangulation with a routing ratio of $9 \cdot c_{dfs} \approx 45.749$. Bose et al. [4] further improved this bound to ≈ 15.479. Then, Bonichon et al. [2] showed that we can locally route on the Delaunay triangulation with a routing ratio of at most 5.9. In the same paper it was shown that the routing ratio of any deterministic local algorithm is at least 1.70 for the Delaunay triangulation.

Efforts to evaluate the spanning ratio and routing ratio have been made for Delaunay triangulations defined on other metrics. We can define these metrics by taking a convex shape and translating and scaling it until it intersects three vertices but contains no points of P in its interior. When we use a circle we obtain the L_2, or classical Delaunay triangulation. When the metric is not specified (as in the rest of this paper), then we are referring to the L_2-Delaunay triangulation. The L_1-Delaunay triangulation uses an axis aligned square, while the L_∞-Delaunay triangulation uses a square tipped at 45 degrees. By rotating the point set 45 degrees, it is easy to show that the L_1 and L_∞ triangulations are equivalent. Bonichon et al. [3] showed that the L_1 and L_∞ Delaunay triangulations are $\sqrt{4+2\sqrt{2}} \approx 2.61$-spanners, and they showed that this bound is tight. On this triangulation, Chew [7] proposed a routing algorithm with routing ratio $\sqrt{10}$. Moreover, the routing ratio of any deterministic local algorithm is at least 2.70 for this class of graph [1]. The TD-Delaunay triangulation is constructed using an equilateral triangle. Chew [8] showed that they are 2-spanners. Bose et al. [5] proposed a routing algorithm of routing ratio $\sqrt{5/3} \approx 2.89$ and they show that this ratio is the best possible. Recently Dennis, Perkovic and Duru [9] showed that the stretch factor of $Hexagon$-Delaunay triangulation is 2 and this is tight.

In this paper we present a local routing algorithm, called *MixedChordArc*, for the L_2-Delaunay triangulation, with a routing ratio of 3.56. This improves the current best routing ratio of 5.9 [1]. Table 1 shows our result in the context of spanning and routing ratios of other Delaunay triangulations.

In Section 2 we define a local algorithm that achieves this routing ratio. In Section 3 we prove the result for a special case, called *balanced configurations*. In Section 4 we extend the technique presented in Section 3 to prove the main result in the general case. In Section 5 we present our conclusions and our ideas for future directions for this line of research.

2 The MixedChordArc Algorithm

Let P be a finite set of points in the plane, and let $DT(P)$ be the Delaunay triangulation of P. We want to route a packet between two vertices of P along edges of $DT(P)$ using only local knowledge and knowledge of our start and destination vertices.

Let s and t be the start and terminal vertices respectively, and assume, without loss of generality, that s and t are on the x-axis with s to the left of t. Our general position assumption ensures that no other vertex lies on st. Consider two triangles T and T' whose interior is cut by st. We say that T is to the left of T', and T' is to the right of T, if, by following st starting at s we intersect T before T'. If uv is the edge shared by T and T', then our general position assumption ensures that u and v are on opposite sides of st.

Let C be a circle that intersects st. We denote by t_C the rightmost point of C on st. Let u and v be two points on C. We denote by $\mathcal{A}_C(u,v)$ the clockwise arc of C from u to v, and by $\mathcal{B}_C(u,v)$ the counter-clockwise arc of C from u to v. We denote the length of a continuous curve S by $|S|$.

Let $p \neq t$ be the vertex representing the current location of the packet. We assume s to be above st, and we assume t to be on the opposite side of st from the current vertex. Let T be the rightmost triangle with p as a vertex whose interior is cut by st. Let $a \neq p$ be the vertex of T that is above st, and let $b \neq p$ be the vertex of T that is below st. Let C be the circumcircle of T.

Here is the algorithm MixedChordArc. First assume that $p = s$. If $|\mathcal{A}_C(s,t_C)| \leq |\mathcal{B}_C(s,t_C)|$, set $p = a$, otherwise set $p = b$. See Fig. 1a. If $p \neq s$, we repeat the following until $p = t$.

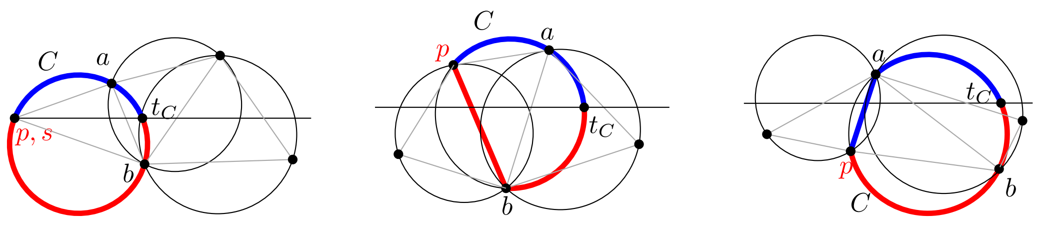
1. If p is above st:
   a. If $|\mathcal{A}_C(p,t_C)| \leq |pb| + |\mathcal{B}_C(b,t_C)|$, set $p = a$
   b. Else set $p = b$.
2. If p is below st:
   a. If $|\mathcal{B}_C(p,t_C)| \leq |pa| + |\mathcal{A}_C(a,t_C)|$, set $p = b$
   b. Else set $p = a$.

Note that assuming that t is on the opposite side of st from p ensures that when t is a neighbour of the current vertex, the algorithm will forward the packet directly to t.

The possible choices are illustrated in Fig. 1. Let $\mathcal{P}\langle s,t\rangle = (s = p_0, p_1, ..., p_n = t)$ be the sequence of vertices produced by the algorithm. In this paper we prove the following theorem.

▶ **Theorem 1.** *The MixedChordArc Algorithm finds a path $\mathcal{P}\langle s,t\rangle$ from s to t whose length $|\mathcal{P}\langle s,t\rangle|$ is not more than $\mu|st|$, where $\mu = \sqrt{\frac{2}{1-\sin(1)}} < 3.56$.*

In some cases, the path produced by our algorithm is a *balanced configuration*. In such cases, the analysis of the length of $\mathcal{P}\langle s,t\rangle$ is much easier. In Section 3 we define what a balanced configuration is, and analyze the length of $\mathcal{P}\langle s,t\rangle$ for this specific case. Then, in Section 4, we analyze the length of $\mathcal{P}\langle s,t\rangle$ for the general case.



(a) From $p = s$, the blue arc is shorter than the red arc, so we forward to a.

(b) From p, the blue path is shorter than the red path, so we forward to a.

(c) From p, the blue path is shorter than the red path, so we forward to a.

Figure 1 Illustrating one step of the algorithm.

3 Bounding $|\mathcal{P}\langle s,t\rangle|$ in a Balanced Configuration

Let us consider a path $\mathcal{P}\langle s,t\rangle$ of vertices such that $p_0 = s, p_n = t$ and $p_{i-1}p_i$ is an edge of the rightmost triangle T_i of p_{i-1} that has a non-empty intersection with st. Let a_i and b_i be the other two vertices of T_i, where a_i is above st, and b_i is below st. Thus $p_i = a_i$ or $p_i = b_i$. Let $s = p_0 = a_0 = b_0$ and let $t = p_n = a_n = b_n$. Let C_i be the circumcircle of T_i, let r_i be its radius and let c_i be its center. Let C_0 be the circle centered at s with radius $r_0 = 0$. Let $\mathcal{T} = (T_1, T_2, ..., T_n)$, and let $\mathcal{C} = (C_0, C_1, ..., C_n)$ be the sequence of circles starting at C_0, followed by the circumcircles of $\mathcal{T}$. Note that the vertex of T_i that is on the opposite side of st to p_{i-1} may not be at the intersection of C_{i-1} and C_i. Thus we define a second intersection point of C_{i-1} and C_i as follows (p_{i-1} being one intersection point). If p_{i-1} is above st, then q_i is the lowest intersection of C_i and C_{i-1} (where "lowest" is defined by the point having the least y-coordinate). If p_{i-1} is below st, let q_i be the highest intersection of C_{i-1} and C_i (where "highest" is defined by the point having the greatest y-coordinate). Note that it is possible to have C_{i-1} and C_i intersect in two points, and still have $q_i = p_{i-1}$. See circle C_4 in Fig. 2. Observe that if T_i and T_{i-1} share an edge, then q_i is the vertex of T_i on the opposite side of st from p_{i-1}. See circles C_1, C_2, C_3, and C_5 in Fig. 2. To simplify the notation, we write t_i instead of t_{C_i}, and we write $\mathcal{A}_i(u,v)$ and $\mathcal{B}_i(u,v)$ instead of $\mathcal{A}_{C_i}(u,v)$ and $\mathcal{B}_{C_i}(u,v)$, respectively.

We say that a pair of consecutive circles C_{i-1} and C_i is *balanced* if $|\mathcal{A}_i(p_{i-1},t_i)| = |p_{i-1}q_i| + |\mathcal{B}_i(q_i,t_i)|$ when p_{i-1} is above st, and if $|\mathcal{B}_i(p_{i-1},t_i)| = |p_{i-1}q_i| + |\mathcal{A}_i(q_i,t_i)|$ when p_{i-1} is below st. A path $\mathcal{P}\langle s,t\rangle$ on a point set P is a *balanced configuration* when C_{i-1} and C_i are balanced for all $1 \le i \le n$.

3.1 Analysis Technique

▶ **Lemma 2.** *Let C_{i-1} and C_i be arbitrary circles of $\mathcal{C}$, where $1 \le i \le n$. Then*
1. $|p_{i-1}b_i| + |\mathcal{B}_i(b_i,t_i)| \le |p_{i-1}q_i| + |\mathcal{B}_i(q_i,t_i)|$ *when p_{i-1} is above st, and*
2. $|p_{i-1}a_i| + |\mathcal{A}_i(a_i,t_i)| \le |p_{i-1}q_i| + |\mathcal{A}_i(q_i,t_i)|$ *when p_{i-1} is below st.*

Proof. By the triangle inequality we have $|p_{i-1}b_i| \le |p_{i-1}q_i| + |\mathcal{B}_i(q_i,b_i)|$, from which 1 follows. Case 2 is symmetric. ◀

For the rest of this section, we assume that $\mathcal{P}\langle s,t\rangle$ is a balanced configuration. Consider the case when p_{i-1} is above st (the case when p_{i-1} is below st is symmetric). If $q_i = b_i$ then $|\mathcal{A}_i(p_{i-1},t_i)| = |p_{i-1}b_i| + |\mathcal{B}_i(b_i,t_i)|$, and the algorithm proceeds to a_i. If $q_i \ne b_i$, observe that $|p_{i-1}b_i| \le |p_{i-1}q_i| + |\mathcal{B}_i(q_i,b_i)|$ by the triangle inequality (see circles C_4 and C_5 in Fig. 2). Thus we have $|p_{i-1}b_i| + |\mathcal{B}_i(b_i,t_i)| < |p_{i-1}q_i| + |\mathcal{B}_i(q_i,t_i)| = |\mathcal{A}_i(p_{i-1},t_i)|$, and the algorithm

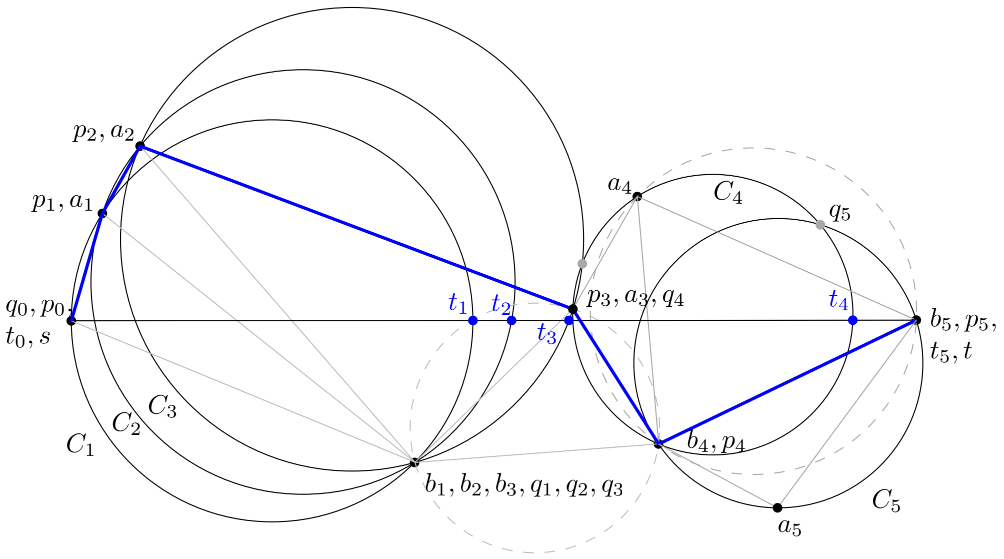


Figure 2 Sequence of circles in a balanced configuration and the path in blue. The dotted circles are circumcircles of triangles intersected by st but not in $\mathcal{T}$.

proceeds to b_i. Thus a balanced configuration allows for steps that cross st and steps that do not cross st. It also allows us to use $|\mathcal{A}_i(p_{i-1}, t_i)|$ as an upper bound on $|p_{i-1}b_i| + |\mathcal{B}_i(b_i, t_i)|$ in the case where $p_{i-1}p_i$ crosses st.

Let $x(v)$ and $y(v)$ be the x and y-coordinates of a point v, respectively. Let s_i be a point on st such that $x(s_i) = x(t_i) - 2r_i$. We define the following potential function that we use to bound the length of $\mathcal{P}\langle s, t\rangle$.

▶ **Definition 3.** If p_{i-1} is above st, then

$$\Phi(C_{i-1}, C_i) = |\mathcal{A}_i(p_{i-1}, t_i)| - |\mathcal{A}_{i-1}(p_{i-1}t_{i-1})| - \lambda|s_{i-1}s_i| - (\mu - \lambda)|t_{i-1}t_i|.$$

Otherwise, if p_{i-1} is below st, then

$$\Phi(C_{i-1}, C_i) = |\mathcal{B}_i(p_{i-1}, t_i)| - |\mathcal{B}_{i-1}(p_{i-1}t_{i-1})| - \lambda|s_{i-1}s_i| - (\mu - \lambda)|t_{i-1}t_i|,$$

where $\lambda = \left(\frac{1+\sin(1)}{\cos(1)} - \pi/2 - 1\right)/2 \approx 0.42$ and $\mu = \sqrt{\frac{2}{1-\sin(1)}} < 3.56$.

See Fig. 2 and 3 for a complete example and an illustration of the potential functions. See Fig. 4 for an illustration of $\Phi(C_{i-1}, C_i)$. Three lemmas are used to prove Theorem 1 for balanced configurations. The proof of Lemma 4 is found in Section 3.3 while the proof of Lemma 5 is in Section 3.2.

▶ **Lemma 4.** *Given a pair of balanced circles C_{i-1} and C_i,*

$$\Phi(C_{i-1}, C_i) \leq 0.$$

▶ **Lemma 5.** *For any balanced configuration $\mathcal{P}\langle s, t\rangle$, $\sum_{i=1}^{n} |s_{i-1}s_i| \leq |st|$.*

▶ **Lemma 6.** *For any $\mathcal{C}$, $x(t_{i-1}) < x(t_i)$ for all $1 \leq i \leq n$, and $\sum_{i=1}^{n} |t_{i-1}t_i| \leq |st|$.*

Proof. We prove that $x(t_{i-1}) < x(t_i)$, that is, t_i is right of t_{i-1} for all $1 \leq i \leq n$, by contradiction. Assume that $x(t_{i-1}) \geq x(t_i)$. If q_i is to the same side of st as p_{i-1}, then C_{i-1} must contain the vertex of T_i on the opposite side of st. If q_i is on the opposite side of

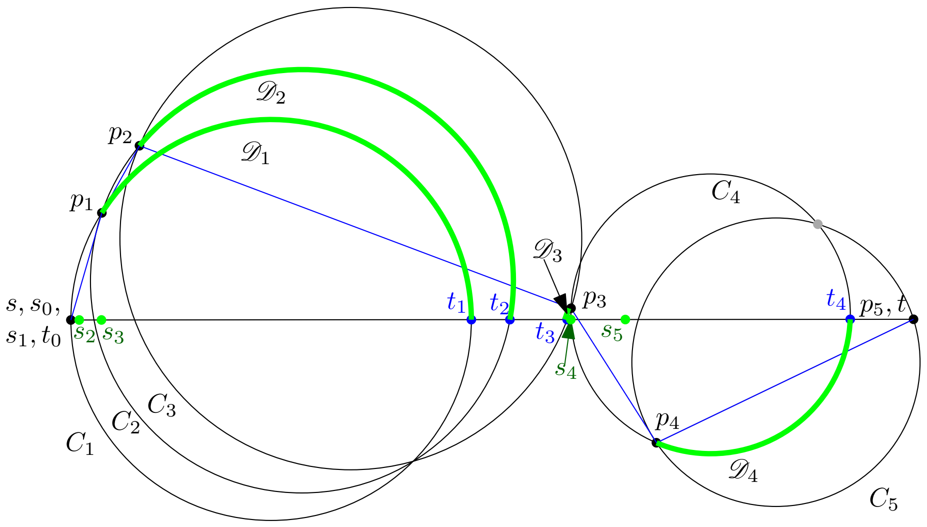


Figure 3 Illustrating the non-zero potential functions $\mathcal{D}_i, 1 \leq i \leq 4$ of a balanced configuration.

st as p_{i-1}, then C_{i-1} contains the vertex of T_i on the same side of st as p_{i-1}. Both cases contradict the construction of a Delaunay triangulation. This, together with the fact that $t_0 = s$ and $t_n = t$ implies the second part of the lemma. ◀

▶ **Lemma 7.** *For $1 \leq i \leq n$, if p_{i-1} is above st, then*
1. **a.** $|\mathcal{A}_i(p_{i-1}, t_i)| > |p_{i-1}p_i| + |\mathcal{A}_i(p_i, t_i)|$ *if p_i is above st, and*
   b. $|\mathcal{A}_i(p_{i-1}, t_i)| > |p_{i-1}p_i| + |\mathcal{B}_i(p_i, t_i)|$ *if p_i is below st*
   otherwise p_{i-1} is below st and
2. **a.** $|\mathcal{B}_i(p_{i-1}, t_i)| > |p_{i-1}p_i| + |\mathcal{B}_i(p_i, t_i)|$ *if p_i is below st, and*
   b. $|\mathcal{B}_i(p_{i-1}, t_i)| > |p_{i-1}p_i| + |\mathcal{A}_i(p_i, t_i)|$ *if p_i is above st.*

Proof. Case 1a is because $|\mathcal{A}_i(p_{i-1}, p_i)| > |p_{i-1}p_i|$, and Case 1b is because if p_i is below st, then the algorithm chose to cross st, which implies 1b. Case 2 is symmetric. ◀

Theorem 1 follows from Lemmas 4, 5, 6, and 7:

Proof. We first analyze the case when p_{i-1} is above st. Recall that in this case, $\Phi(C_{i-1}, C_i)$ is defined as

$$\Phi(C_{i-1}, C_i) = |\mathcal{A}_i(p_{i-1}, t_i)| - |\mathcal{A}_{i-1}(p_{i-1}t_{i-1})| - \lambda|s_{i-1}s_i| - (\mu - \lambda)|t_{i-1}t_i|.$$

If p_i is above st (same side of st as p_{i-1}), then $|\mathcal{A}_i(p_{i-1}, t_i)| > |p_{i-1}p_i| + |\mathcal{A}_i(p_i, t_i)|$ by Lemma 7. In this case, let $\mathcal{D}_i = \mathcal{A}_i(p_i, t_i)$. If p_i is below st, then $|\mathcal{A}_i(p_{i-1}, t_i)| > |p_{i-1}p_i| + |\mathcal{B}_i(p_i, t_i)|$ by Lemma 7. In this case, let $\mathcal{D}_i = \mathcal{B}_i(p_i, t_i)$. In both cases we have $|\mathcal{A}_i(p_{i-1}, t_i)| > |p_{i-1}p_i| + |\mathcal{D}_i|$.

Let $\Phi'(C_{i-1}, C_i)$ be the function defined by

$$\Phi'(C_{i-1}, C_i) = |p_{i-1}p_i| + |\mathcal{D}_i| - |\mathcal{D}_{i-1}| - \lambda|s_{i-1}s_i| - (\mu - \lambda)|t_{i-1}t_i|.$$

Observe that $\Phi'(C_{i-1}, C_i) \leq \Phi(C_{i-1}, C_i)$. By Lemma 4, $\Phi(C_{i-1}, C_i) \leq 0$, thus $\Phi'(C_{i-1}, C_i) \leq 0$. When p_{i-1} is below st, a symmetric proof again shows us that $\Phi'(C_{i-1}, C_i) \leq 0$. Recall

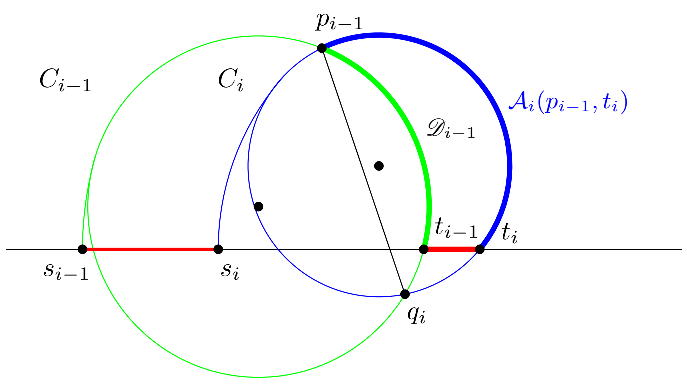


Figure 4 Illustrating the function $\Phi(C_{i-1}, C_i)$: blue minus green is charged to red to obtain an upper bound on the routing ratio.

that $p_0 = t_0 = s$, and $p_n = t_n = t$, which means $|\mathcal{D}_0| = |\mathcal{D}_n| = 0$. Therefore we have

$$\sum_{i=1}^{n} \Phi'(C_{i-1}, C_i) \leq 0$$

from which we get:

$$\sum_{i=1}^{n} (|p_{i-1}p_i| + |\mathcal{D}_i| - |\mathcal{D}_{i-1}|) \leq \sum_{i=1}^{n} (\lambda |s_{i-1}s_i| + (\mu - \lambda)|t_{i-1}t_i|)$$

$$|\mathcal{P}\langle s,t\rangle| - |\mathcal{D}_0| + |\mathcal{D}_n| \leq (\lambda + \mu - \lambda)|st| \tag{1}$$

$$|\mathcal{P}\langle s,t\rangle| \leq \mu |st|.$$

The right hand side of (1) is due to Lemmas 5 and 6. ◀

Lemma 5 is discussed in the next section. Lemma 4 is discussed in Section 3.3.

3.2 Proof of Lemma 5

Lemma 5 uses the following supporting result:

▶ **Lemma 8.** *Let C_{i-1} and C_i be balanced. Let s_{i-1} be the point on st where $x(s_{i-1}) = x(t_{i-1}) - 2r_{i-1}$ and let s_i be the point on st where $x(s_i) = x(t_i) - 2r_i$. Then $x(s_{i-1}) \leq x(s_i)$.*

Proof. See Fig. 5. Let u_{i-1} be the point on C_{i-1} that is diametrically opposed to t_{i-1} and let u_i be the point on C_i that is diametrically opposed to t_i. We will show the case when p_{i-1} is above st; the case when it is below st is symmetric. Since C_{i-1} and C_i are balanced, we have that $|\mathcal{A}_i(p_{i-1}, t_i)| = |p_{i-1}q_i| + |\mathcal{B}_i(q_i, t_i)|$ which implies that $|\mathcal{A}_i(p_{i-1}, t_i)| \leq \pi r_i$ and $|\mathcal{B}_i(q_i, t_i)| \leq \pi r_i$. Since $|\mathcal{A}_i(u_i, t_i)| = |\mathcal{B}_i(u_i, t_i)| = \pi r_i$, u_i is not on the open interval $\mathcal{A}_i(p_{i-1}, t_i)$ or $\mathcal{B}_i(q_i, t_i)$, which implies that either u_i is on the arc of C_i between p_{i-1} and q_i that does not contain t_i, or $u_i = p_{i-1} = q_i$. Lemma 6 implies that t_i is not inside C_{i-1}, which implies that u_i must be on or inside C_{i-1}. Let O_i be the circle centered at t_i with radius $|t_i u_i| = 2r_i$. Thus O_i and C_i are tangent at u_i, and O_i intersects st at s_i. Let O_{i-1} be the circle centered at t_{i-1} with radius $2r_{i-1}$. Thus O_{i-1} and C_{i-1} are tangent at u_{i-1}, and O_{i-1} intersects st at s_{i-1}. We prove the lemma by contradiction, thus assume that $x(s_i) < x(s_{i-1})$. In the proof of Lemma 6, we showed that $x(t_i) > x(t_{i-1})$. Therefore, it must be that O_{i-1} is in the interior of O_i, and thus they do not intersect. Since u_i is on or

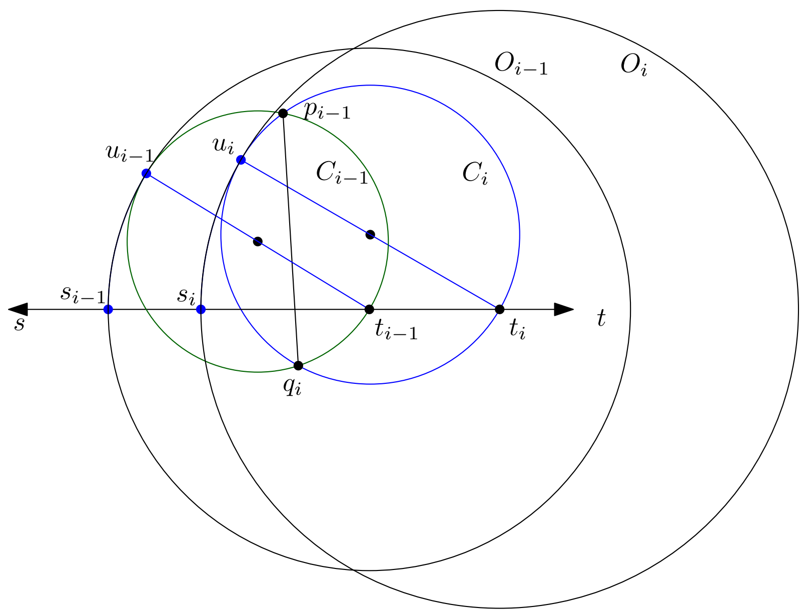


Figure 5 O_i must intersect O_{i-1} if C_{i-1} and C_i are path balanced, which implies that $x(s_{i-1}) \leq x(s_i)$.

inside C_{i-1}, and O_i intersects u_i, O_i must intersect C_{i-1}. But C_{i-1} is contained in O_{i-1} except for the point u_{i-1}, and O_{i-1} is contained in O_i, and thus O_i cannot intersect C_{i-1}, which is a contradiction. See Fig. 5. ◀

We can now prove Lemma 5:

Proof of Lemma 5. Follows from Lemma 8 and the fact that $x(s_0) = x(s)$ and $x(s_n) < x(t)$. ◀

3.3 Proof of Lemma 4

To show that $\Phi(C_{i-1}, C_i) \leq 0$ when C_{i-1} and C_i are balanced, we set up the following coordinate system. We show the proof for the case when p_{i-1} is above st; the case when p_{i-1} is below st is symmetric. Let c_{i-1} and c_i lie along the x-axis, and let p_{i-1} and q_i lie along the y-axis. See Fig. 6. Lemma 4 follows from the following two lemmas:

▶ **Lemma 9.** *When C_{i-1} and C_i are balanced, if $y(t_{i-1}) \leq 0$, then $\Phi(C_{i-1}, C_i) \leq 0$.*

▶ **Lemma 10.** *When C_{i-1} and C_i are balanced, if $y(t_{i-1}) > 0$, then $\Phi(C_{i-1}, C_i) \leq 0$.*

The main tool to prove these two lemmas is the following transformation, which is similar to a transformation used by Xia [13].

▶ **Transformation 11.** *Fix p_{i-1} and q_i, and translate c_i to the left along the x-axis until $c_i = c_{i-1}$. Moreover keep C_{i-1} unchanged and maintain C_i as the circle with center c_i with p_{i-1} on its boundary.*

Observe that, after we have completed Transformation 11, we have $C_i = C_{i-1}$ and thus $\Phi(C_{i-1}, C_i) = 0$. If we can show that $\Phi(C_{i-1}, C_i)$ is increasing while $x(c_i)$ decreases, then it must be that $\Phi(C_{i-1}, C_i) \leq 0$ before Transformation 11. Thus we wish to find the change in $\Phi(C_{i-1}, C_i)$ with respect to the change in $x(c_i)$ during Transformation 11. Formally:

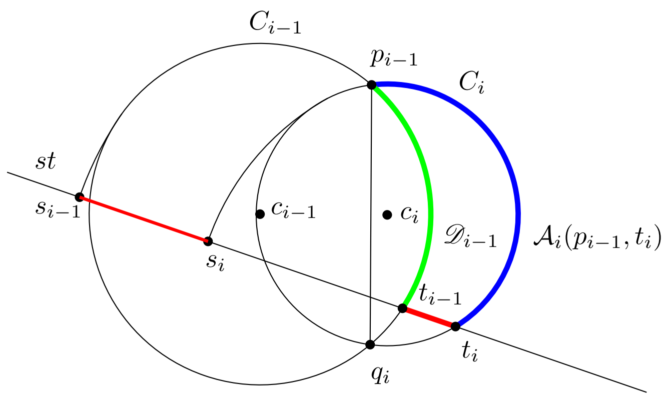


Figure 6 Coordinate system for analyzing $\Phi(C_{i-1}, C_i)$.

▶ **Lemma 12.** *If* $\frac{d\Phi(C_{i-1},C_i)}{dx(c_i)} \leq 0$ *during Transformation 11, then* $\Phi(C_{i-1}, C_i) \leq 0$.

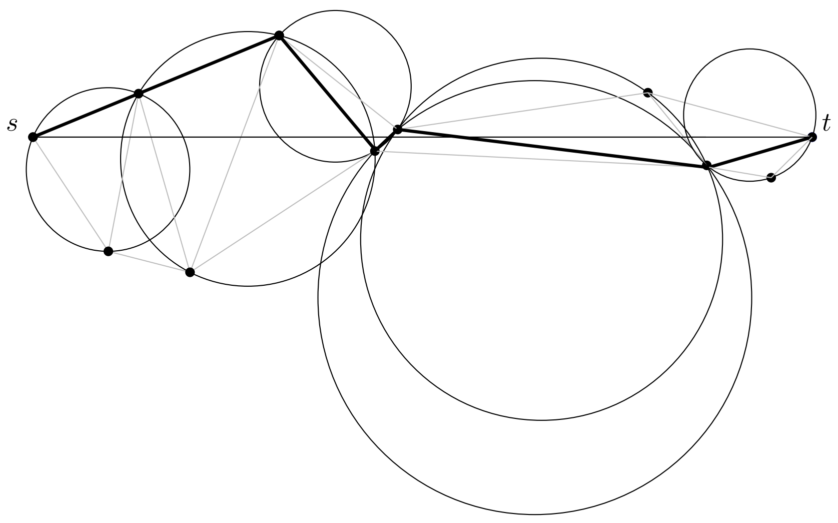
Proof. At the end of Transformation 11 we have that $\Phi(C_{i-1}, C_i) = 0$. If $\frac{d\Phi(C_{i-1},C_i)}{dx(c_i)} \leq 0$ then $\Phi(C_{i-1}, C_i)$ is not decreasing during Transformation 11, and thus $\Phi(C_{i-1}, C_i) \leq 0$ before Transformation 11. ◀

The analysis of this function is similar to Xia's approach [13]. To ensure that this transformation is well-defined, we require q_i to be below st. We observe that $\Phi(C_{i-1}, C_i)$ is maximized when st is on or above c_{i-1}, and this assumption implies q_i is below st (or on st, in the case where $p_{i-1} = q_i$). Full details of this analysis, the transformation analysis, and the proofs for Lemmas 9 and 10 have been left out due to space constraints.

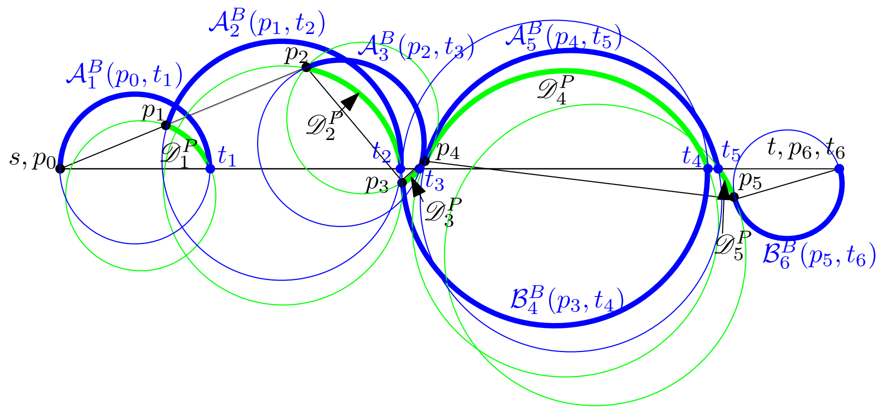
4 Bounding $\mathcal{P}\langle s, t\rangle$ in the General Case

In Section 3, we proved Theorem 1 for the case when the path produced by our algorithm results in a balanced configuration. In this section, we prove Theorem 1 for the general case. Given a sequence $\mathcal{C}$ of circles that intersect st, no series of transformations were found that could achieve a balanced configuration, while simultaneously providing a provable upper bound on the length of $|p_{i-1}, p_i|$. However, we were able to find *two* sequences of circles to substitute for $\mathcal{C}$. To represent each C_i in $\mathcal{C}$, we have a *potential circle* C_i^P and a *bounding circle* C_i^B. Like C_i, both C_i^P and C_i^B have t_i as their rightmost intersection with st. However, C_i intersects both p_i and p_{i-1}, while C_i^B is only required to intersect p_{i-1}, and C_i^P is only required to intersect p_i. If we look at a bounding circle C_i^B and the previous potential circle C_{i-1}^P, which intersect at p_{i-1}, they are balanced, and we can thus apply the function $\Phi(C_{i-1}^P, C_i^B)$ to relate the lengths of the arcs of these circles to $|st|$. Finally, when analyzed properly, they provide an upper bound on the length $|p_i p_{i-1}|$.

Formally, let C_0^P be the circle centered at $s = p_0$ with radius $r_0^P = 0$, and let C_n^P be the circle centered at t with radius $r_n^P = 0$. Assuming we have defined C_{i-1}^P, we will define C_i^B and C_i^P. If p_{i-1} is above st, let C_i^B be the circle through p_{i-1} and t_i for which $|\mathcal{A}_{C_i^B}(p_{i-1}, t_i)| = |p_{i-1}q_i'| + |\mathcal{B}_{C_i^B}(q_i', t_i)|$, where q_i' is the bottommost intersection of C_{i-1}^P and C_i^B. If p_{i-1} is below st, let C_i^B be the circle through p_{i-1} and t_i for which $|\mathcal{B}_{C_i^B}(p_{i-1}, t_i)| = |p_{i-1}q_i'| + |\mathcal{A}_{C_i^B}(q_i', t_i)|$, where q_i' is the topmost intersection of C_{i-1}^P and C_i^B. That is, C_{i-1}^P and C_i^B are balanced. Let r_i^B be the radius of C_i^B. The potential circle C_i^P is the circle through p_i, whose rightmost intersection with st is t_i, and whose radius is



(a) The sequence of triangles $\mathcal{T}$ intersected by st, along with their circumcircles $\mathcal{C}$, and the path $\mathcal{P}\langle s,t\rangle$ found by the algorithm in bold.



$\mathcal{A}_2^B(p_1,t_2)$
$\mathcal{A}_3^B(p_2,t_3)$
$\mathcal{A}_5^B(p_4,t_5)$
$\mathcal{A}_1^B(p_0,t_1)$
p_2
$\mathscr{D}_4^P$
p_1
$\mathscr{D}_2^P$
$\mathscr{D}_1^P$
t_1
s,p_0
t_2
p_4
t_4
t_5
t,p_6,t_6
p_3
t_3
p_5
$\mathscr{D}_3^P$
$\mathscr{D}_5^P$
$\mathcal{B}_6^B(p_5,t_6)$
$\mathcal{B}_4^B(p_3,t_4)$

(b) The complete set of bounding arcs and potential arcs used in the function $\Phi(C_{i-1}^P, C_i^B)$, used to bound the routing ratio in the general case.

Figure 7 The initial circumcircles in 7a, and the construction of the potential circles and bounding circles in the general case in 7b.

given by $r_i^P = \min\{r_i, r_i^B\}$ (with the exception of $r_n^P = 0$). Let s_i^P be the point on st with $x(s_i^P) = x(t_i) - 2r_i^P$, and let s_i^B be the point on st with $x(s_i^B) = x(t_i) - 2r_i^B$.

To simplify notation, for points u and v on C_i^P, instead of writing $\mathcal{A}_{C_i^P}(u,v)$ and $\mathcal{B}_{C_i^P}(u,v)$ to indicate clockwise and counter-clockwise arcs of C_i^P from u to v, respectively, we write $\mathcal{A}_i^P(u,v)$ and $\mathcal{B}_i^P(u,v)$. Likewise, for points u and v on C_i^B, instead of writing $\mathcal{A}_{C_i^B}(u,v)$ and $\mathcal{B}_{C_i^B}(u,v)$, we write $\mathcal{A}_i^B(u,v)$ and $\mathcal{B}_i^B(u,v)$.

See Figs. 7a and 7b for an example of the initial sequences $\mathcal{T}$ and $\mathcal{C}$ and the resulting bounding and potential arcs that we are interested in.

Since C_{i-1}^P and C_i^B are balanced, Φ can be extended to C_{i-1}^P and C_i^B, and thus we have

$$\Phi(C_{i-1}^P, C_i^B) = |\mathcal{A}_i^B(p_{i-1},t_i)| - |\mathcal{A}_{i-1}^P(p_{i-1},t_{i-1})| - \lambda|s_{i-1}^P s_i^B| - \mu|t_{i-1}t_i|$$

when p_{i-1} is above st and

$$\Phi(C_{i-1}^P, C_i^B) = |\mathcal{B}_i^B(p_{i-1},t_i)| - |\mathcal{B}_{i-1}^P(p_{i-1},t_{i-1})| - \lambda|s_{i-1}^P s_i^B| - \mu|t_{i-1}t_i|$$

when p_{i-1} is below st. Lemma 4 tells us that $\Phi(C^P_{i-1}, C^B_i) \le 0$. To prove Theorem 1 in the general case, it is sufficient to prove the following two lemmas. Lemma 13 is a generalization of Lemma 5, whereas Lemma 14 is a generalization of Lemma 7.

▶ **Lemma 13.** $\sum_{i=1}^{n} |s^P_{i-1}s^B_i| \le |st|$.

Proof. Since C^P_{i-1} and C^B_i are balanced, Lemma 8 tells us that $x(s^P_{i-1}) \le x(s^B_i)$. We know that $x(s^P_i) = x(t_i) - 2r^P_i$ and $x(s^B_i) = x(t_i) - 2r^B_i$, thus the fact that $r^P_i = \min\{r_i, r^B_i\}$ implies that $x(s^B_i) \le x(s^P_i)$. Thus $|s^P_{i-1}s^B_i| \le |s^P_{i-1}s^P_i|$, and it is sufficient to show that $\sum_{i=1}^{n} |s^P_{i-1}s^P_i| \le |st|$. The fact that $x(s^P_{i-1}) \le x(s^B_i)$ implies that $x(s^P_{i-1}) \le x(s^P_i)$, and C^P_0 is the circle centered at s with radius 0, and thus $s^P_0 = s$. Since $x(s^P_n) \le x(t)$, this completes the proof. ◀

Due to space constraints, we omit the proof of the following lemma.

▶ **Lemma 14.** *For $1 \le i \le n$, if p_{i-1} is above st, then*

1. a. $|\mathcal{A}^B_i(p_{i-1}, t_i)| \ge |p_{i-1}p_i| + |\mathcal{A}^P_i(p_i, t_i)|$ *if p_i is above st, and*

b. $|\mathcal{A}^B_i(p_{i-1}, t_i)| \ge |p_{i-1}p_i| + |\mathcal{B}^P_i(p_i, t_i)|$ *if p_i is below st*

otherwise p_{i-1} is below st and

2. a. $|\mathcal{B}^B_i(p_{i-1}, t_i)| \ge |p_{i-1}p_i| + |\mathcal{B}^P_i(p_i, t_i)|$ *if p_i is below st, and*

b. $|\mathcal{B}^B_i(p_{i-1}, t_i)| \ge |p_{i-1}p_i| + |\mathcal{A}^P_i(p_i, t_i)|$ *if p_i is above st.*

Theorem 1 follows from Lemmas 4, 6, 13, and 14.

Proof of Theorem 1. If p_i is above st, let $\mathcal{D}^P_i = \mathcal{A}^P_i(p_i, t_i)$. If p_i is below st, let $\mathcal{D}^P_i = \mathcal{B}^P_i(p_i, t_i)$. Let $\Phi'(C^P_{i-1}, C^B_i) = |p_{i-1}p_i| + |\mathcal{D}^P_i| - |\mathcal{D}^P_{i-1}| - \lambda|s^P_{i-1}s^B_i| - (\mu - \lambda)|t_{i-1}t_i|$. Lemmas 14 and 4 imply that $\Phi'(C^P_{i-1}, C^B_i) \le \Phi(C^P_{i-1}, C^B_i) \le 0$. Using $\Phi'(C^P_{i-1}, C^B_i)$ we get:

$$\sum_{i=1}^{n} \Phi'(C_{i-1}, C_i) \le 0$$

$$\sum_{i=1}^{n} \left(|p_{i-1}p_i| + |\mathcal{D}^P_i| - |\mathcal{D}^P_{i-1}|\right) \le \sum_{i=1}^{n} (\lambda|s^P_{i-1}s^B_i| + (\mu - \lambda)|t_{i-1}t_i|)$$

$$|\mathcal{P}\langle s, t\rangle| - |\mathcal{D}^P_0| + |\mathcal{D}^P_n| \le (\lambda + \mu - \lambda)|st| \tag{2}$$

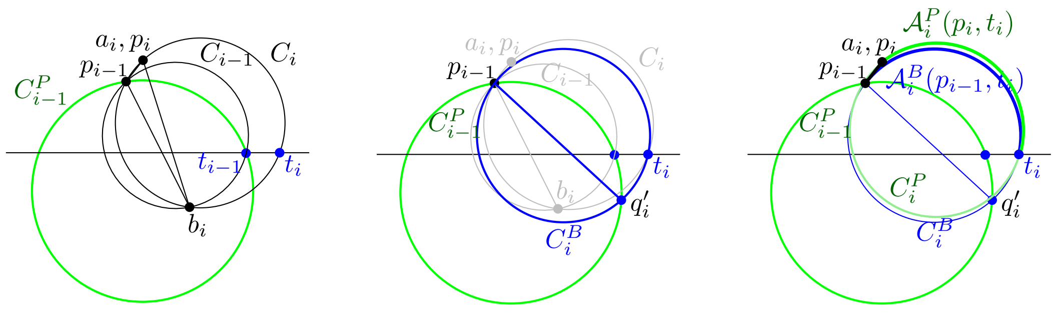
$$|\mathcal{P}\langle s, t\rangle| \le \mu|st|.$$

Line (2) follows from Lemmas 6 and 13. ◀

We give some insight into the selection of r^P_i. Assume that p_{i-1} is above st (when p_{i-1} is below st the explanation is symmetric). The purpose of $|\mathcal{A}^B_i(p_{i-1}, t_i)|$ is to bound $|p_{i-1}p_i| + |\mathcal{A}^P_i(p_i, t_i)|$, as expressed in Lemma 14. This lemma is also the reason for selecting the radius of C^P_i as $r^P_i = \min\{r_i, r^B_i\}$. It would be simpler to let $r^P_i = r^B_i$, since then we would have $s^P_i = s^B_i$. However, if we allow $r^P_i > r_i$, it can happen that the arc $|\mathcal{A}^B_{i+1}(p_i, t_{i+1})|$ on the next bounding circle is not large enough to cover $|p_ip_{i+1}| + |\mathcal{A}^P_{i+1}(p_{i+1}, t_{i+1})|$. See Fig. 8. Thus Lemma 14 would not hold. To account for this, we ensure that C^P_i has radius at most r_i.

5 Conclusion and Future Work

Consider the algorithm presented in Section 2, along with two variations. To keep the algorithms simple, assume we are at a vertex p above st. Otherwise all assumptions are the same as in Section 2.



(a) C_{i-1}, C_i, and C^P_{i-1}. Notice that $r^P_{i-1} > r_{i-1}$.

(b) C^B_i and its intersection with C^P_{i-1}.

(c) $|\mathcal{A}^B_i(p_{i-1}, t_i)| < |p_{i-1}, p_i| + |\mathcal{A}^P_i(p_i, t_i)|$.

Figure 8 The reasoning behind $r^P_i = \min\{r_i, r^B_i\}$. In this diagram, $r^P_i > r_i$, and we show why it is detrimental to our analysis. Notice that $|\mathcal{A}^B_i(p_{i-1}, t_i)| < |p_{i-1}, p_i| + |\mathcal{A}^P_i(p_i, t_i)|$. Thus the arc $\mathcal{A}^B_i(p_{i-1}, t_i)$ of the bounding circle is not long enough to pay for $|p_{i-1}, p_i| + |\mathcal{A}^P_i(p_i, t_i)|$.

A) **BestChord:** If $|pa| + |\mathcal{A}_C(a, t_C)| \leq |pb| + |\mathcal{A}_C(b, t_C)|$ then $p = a$ else $p = b$.
B) **MixedChordArc:** If $|\mathcal{A}_C(p, t_C)| \leq |pb| + |\mathcal{A}_C(b, t_C)|$ then $p = a$ else $p = b$.
C) **MinArc:** If $|\mathcal{A}_C(p, t_C)| \leq \pi r$ then $p = a$ else $p = b$.

The algorithm presented in this paper is *MixedChordArc*. Following the techniques used in [1] we are able to show that the routing ratio of *MinArc* is between 3.20 and 3.96. Since the routing ratio of 3.56 of *MixedChordArc* is better, we do not present the details of *MinArc*.

We suspect that *BestChord* is an improvement on *MixedChordArc*. It seems plausible that we can modify the proofs presented in this paper to obtain the same upper bound for *BestChord* as for *MixedChordArc*, but for now that remains unverified. Whether or not *BestChord* is asymptotically superior to *MixedChordArc*, or whether they are asymptotically the same is still unknown.

Although we have improved the upper bound of the routing ratio on the L_2-Delaunay triangulation, it is not clear how tight our analysis is. The upper bound on the analysis is where our potential function is the weakest. A more clever potential function could lower the routing ratio using a comparable analysis. Or perhaps one of the algorithms above would respond to a completely different style of analysis.

Furthermore, the lower bound on *MixedChordArc* is still the same as the lower bound on routing on the L_2-Delaunay triangulation in general, which is approximately 1.70 [1]. So it seems there is still much room for improvement. The question remains, what other algorithms or analysis can we use to improve the routing ratio of the Delaunay triangulation? And given that the upper and lower bounds on the spanning ratio of the L_2-Delaunay triangulation are 1.998 [13] and 1.5932 [14] respectively, is there a separation of the spanning and routing ratios of the Delaunay triangulation?

References

1 Nicolas Bonichon, Prosenjit Bose, Jean-Lou De Carufel, Ljubomir Perković, and André van Renssen. Upper and lower bounds for online routing on Delaunay triangulations. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 203–214. Springer Berlin Heidelberg, 2015. doi:10.1007/978-3-662-48350-3_18.

2 Nicolas Bonichon, Prosenjit Bose, Jean-Lou De Carufel, Ljubomir Perković, and André van Renssen. Upper and lower bounds for online routing on Delaunay triangulations. *Discrete & Computational Geometry*, 58(2):482–504, Sep 2017. `doi:10.1007/s00454-016-9842-y`.

3 Nicolas Bonichon, Cyril Gavoille, Nicolas Hanusse, and Ljubomir Perković. Tight stretch factors for L_1 and L_∞ Delaunay triangulations. *Computational Geometry*, 48(3):237–250, 2015. `doi:10.1016/j.comgeo.2014.10.005`.

4 Prosenjit Bose, Jean-Lou De Carufel, Stephane Durocher, and Perouz Taslakian. Competitive online routing on Delaunay triangulations. In R. Ravi and Inge Li Gørtz, editors, *Algorithm Theory - SWAT 2014 - 14th Scandinavian Symposium and Workshops, Copenhagen, Denmark, July 2-4, 2014. Proceedings*, volume 8503 of *Lecture Notes in Computer Science*, pages 98–109. Springer, 2014. `doi:10.1007/978-3-319-08404-6_9`.

5 Prosenjit Bose, Rolf Fagerberg, André van Renssen, and Sander Verdonschot. Competitive routing in the half-theta-6-graph. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1319–1328. SIAM, 2012. URL: `http://dl.acm.org/citation.cfm?id=2095116.2095220`.

6 Prosenjit Bose and Pat Morin. Online routing in triangulations. In *Algorithms and Computation*, volume 1741 of *Lecture Notes in Computer Science*, pages 113–122. Springer Berlin Heidelberg, 1999. `doi:10.1007/3-540-46632-0_12`.

7 L. Paul Chew. There is a planar graph almost as good as the complete graph. In *Proceedings of the Second Annual Symposium on Computational Geometry*, SCG '86, pages 169–177, New York, NY, USA, 1986. ACM. `doi:10.1145/10515.10534`.

8 L. Paul Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39(2):205–219, 1989. `doi:10.1016/0022-0000(89)90044-5`.

9 Michael Dennis, Ljubomir Perković, and Duru Türkoglu. The stretch factor of hexagon-Delaunay triangulations. *CoRR*, abs/1711.00068, 2017. `arXiv:1711.00068`.

10 David P. Dobkin, Steven J. Friedman, and Kenneth J. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete & Computational Geometry*, 5(1):399–407, 1990. `doi:10.1007/BF02187801`.

11 J. Mark Keil and Carl A. Gutwin. Classes of graphs which approximate the complete euclidean graph. *Discrete & Computational Geometry*, 7(1):13–28, 1992. `doi:10.1007/BF02187821`.

12 Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks*. Cambridge University Press, New York, NY, USA, 2007.

13 Ge Xia. Improved upper bound on the stretch factor of Delaunay triangulations. In *Proceedings of the Twenty-seventh Annual Symposium on Computational Geometry*, SoCG '11, pages 264–273, New York, NY, USA, 2011. ACM. `doi:10.1145/1998196.1998235`.

14 Ge Xia and Liang Zhang. Toward the tight bound of the stretch factor of Delaunay triangulations. In *Proceedings of the Canadian Conference on Computational Geometry*, CCCG '11, 2011.

On Geometric Prototype and Applications

Hu Ding
Department of Computer Science and Engineering, Michigan State University
East Lansing, USA; and
School of Computer Science and Technology, University of Science and Technology of China
Hefei, China
huding@msu.edu, huding@ustc.edu.cn

Manni Liu
Department of Computer Science and Engineering, Michigan State University
East Lansing, USA
liumanni@msu.edu

Abstract

In this paper, we propose to study a new geometric optimization problem called the "geometric prototype" in Euclidean space. Given a set of patterns, where each pattern is represented by a (weighted or unweighted) point set, the geometric prototype can be viewed as the "average pattern" minimizing the total matching cost to them. As a general model, the problem finds many applications in real-world, such as Wasserstein barycenter and ensemble clustering. The dimensionality could be either constant or high, depending on the applications. To our best knowledge, the general geometric prototype problem has yet to be seriously considered by the theory community. To bridge the gap between theory and practice, we first show that a small core-set can be obtained to substantially reduce the data size. Consequently, any existing heuristic or algorithm can run on the core-set to achieve a great improvement on the efficiency. As a new application of core-set, it needs to tackle a couple of challenges particularly in theory. Finally, we test our method on both image and high dimensional clustering datasets; the experimental results remain stable even if we run the algorithms on core-sets much smaller than the original datasets, while the running times are reduced significantly.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases prototype, core-set, Wasserstein barycenter, ensemble clustering

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.23

Related Version A full version of the paper is available at [21], `https://arxiv.org/abs/1804.09655`.

Acknowledgements The research of this work was supported in part by NSF through grant CCF-1656905 and a start-up fund from Michigan State University. The authors also want to thank the anonymous reviewers for their helpful comments and suggestions for improving the paper.

1 Introduction

Given a set of points in Euclidean space, we can easily use the geometric mean or median point to represent them. However, if they are replaced by a set of point sets where each point set denotes a "pattern", the problem of finding their representation will be much more challenging. We call it the "geometric prototype" problem. Before introducing its formal definition, we need to define the matching cost between two patterns first.

© Hu Ding and Manni Liu;
licensed under Creative Commons License CC-BY

▶ **Definition 1** ($\mathcal{M}(A, B)$)**.** Given two point sets $A = \{a_1, a_2, \cdots, a_k\}$ and $B = \{b_1, b_2, \cdots, b_k\}$ in $\mathbb{R}^d$,

$$\mathcal{M}(A, B) = \min_{\pi \in \Pi} \sum_{j=1}^{k} ||a_j - b_{\pi(j)}||^2 \tag{1}$$

where Π contains all the possible permutations of $\{1, 2, \cdots, k\}$.

$\mathcal{M}(A, B)$ is in fact the problem of geometric matching which can be optimally solved by the Hungarian algorithm [16]. When the dimensionality is constant, a number of efficient approximation algorithms have been developed in past years (see more discussion in Section 1.1).

▶ **Definition 2** (Geometric Prototype)**.** Given a set of point sets $\mathbb{P} = \{P_1, P_2, \cdots, P_n\}$ with each P_i containing k points $\{p^i_1, p^i_2, \cdots, p^i_k\} \subset \mathbb{R}^d$, the geometric prototype is a new point set $g(\mathbb{P})$ having k points such that

$$\sum_{i=1}^{n} \mathcal{M}(P_i, g(\mathbb{P})) \tag{2}$$

is minimized. Note that $g(\mathbb{P})$ can be a new pattern not from $\mathbb{P}$. Also, any k-point set achieving at most c times the minimum value of (2) is called a c-approximation with $\forall c \geq 1$.

▶ Remark. It is easy to see that when $k = 1$, the geometric prototype is simply the mean point. Actually, the problem of geometric prototype can be viewed as a "chromatic k-means clustering". The kn points of $\cup_{i=1}^{n} P_i$ form k clusters where the k points of each P_i should be assigned to the k clusters separately; to minimize the objective function (2), the k points of $g(\mathbb{P})$ should be the mean points of the resulting clusters.

In Definition 2, the dimension d could be either constant or high depending on the applications, and n usually is large (k could be not constant, but often much smaller than n in the applications). To our best knowledge, the general geometric prototype problem has never been systematically studied in the area of computational geometry (except some special cases; see Section 1.1), but finds many real-world applications recently. Below, we introduce two important applications in low and high dimension, respectively.

(1) Wasserstein Barycenter. Given a large set of images, finding their average yields several benefits in practice. For example, if all the images are taken from the same object but have certain extents of noise, their average image could serve as a robust pattern to represent them; also, this is an efficient way to compress large image datasets. In computer vision, Earth Mover's Distance (EMD) [39] is widely used to measure the difference between two images; the average image minimizing the total EMDs to all the images is defined as the *Wasserstein Barycenter* [9, 10, 17, 29, 46]. In addition, Ding and Xu [20, 24] considered the case allowing rigid/affine transformations for each image. Wasserstein Barycenter can also be applied to Bayesian inference [43]. Note that the geometric prototype defined above is not exactly equivalent to Wasserstein Barycenter, because the latter one requires each point having a non-negative weight and EMD is to minimize the max flow cost; however, the techniques proposed in this paper can be easily extended to handle EMD and we will discuss it later.

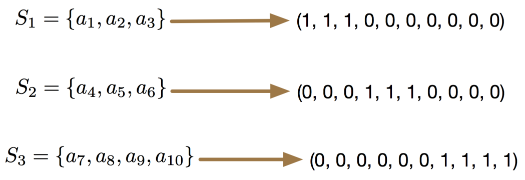


Figure 1 $d = 10$ and $k = 3$. The three clusters are mapped to 3 binary vectors in $\mathbb{R}^{10}$.

(2) Ensemble Clustering. Given a number of different clustering solutions for the same set of items, the problem about finding a unified clustering solution minimizing the total differences to them is called *ensemble clustering* [28]. This problem has attracted a great deal of attention, especially for the applications in big data and crowdsourcing [27,35,42,44]. For example, due to the proliferation of networked sensing systems, we can use a large number of sensors to record the same environment and each sensor can generate an individual clustering for the same set of objects. However, most of existing approaches rely on algebraic or graphic models and need to solve complicated optimizations with high complexities (such as semi-definite programming [42]).

Recently, Ding et al. [22] presented a novel high dimensional geometric model for the problem of ensemble clustering: suppose there are d items and each clustering solution has k clusters on these items (if less than k, we can add some dummy empty clusters); then, each single cluster is mapped to a binary vector in $\mathbb{R}^d$ where each dimension indicates the membership of an individual item (see Figure 1); so each clustering solution is mapped to a k-point set in $\mathbb{R}^d$; the size of the symmetric difference between two clusters is equal to their squared distance in $\mathbb{R}^d$, and thus the difference between two clustering solutions is always equal to half of their matching cost (Definition 1) in Euclidean space. Therefore, finding the final clustering solution minimizing the total differences to the given solutions is equivalent to computing the geometric prototype of the resulting k-point sets in $\mathbb{R}^d$. Please find more details in [22]. Note that the obtained geometric prototype may result in fractional clustering memberships, because the points of the geometric prototype are not necessarily binary vectors. So the approximation result in [22] does not violate the APX-hardness for strict ensemble/consensus clustering [11]. Actually fractional clustering memberships are acceptable and make sense in practice; for instance, we may claim that one object belongs to class 1, 2, and 3 with probabilities of 70%, 20%, and 10%, respectively.

1.1 Our Main Contributions and Related Work

Due to the non-convex nature of the geometric prototype problem, most of the aforementioned approaches for Wasserstein barycenter [9,10,17,29,46] and large-scale ensemble clustering [22] are iterative algorithms, such as alternating minimization and Alternating Direction Method of Multipliers (ADMM) [12], which can converge to some local optimums. Those approaches could be very slow for large datasets, because they may run many rounds and each round usually needs to conduct some complicated update or optimization. This is also the main motivation of our work, that is, replacing the original large input by a small core-set to speed up the computation of existing algorithms.

In this paper, our contribution is twofold in the aspects of theory and applications. In theory, we show that a small core-set can be obtained for the problem of geometric prototype. More importantly, our core-set is independent of any geometric prototype algorithm; namely, we can run any available algorithm as a black box on the core-set, instead of the original instance $\mathbb{P}$, to achieve a similar result. Although core-set has been extensively studied for many applications before [3, 38], we still need to tackle several significant challenges when constructing the core-set for geometric prototype. In practice, we test our method for solving the applications Wasserstein barycenter and ensemble clustering. The experiment shows that running the existing algorithms on core-sets can achieve almost the same results while the running times are substantially reduced.

Related work. The general geometric prototype problem has yet to be seriously considered by the theory community (to our best knowledge), however, some special cases were studied before. Based on the remark below Definition 2, we know that finding the geometric prototype is also a chromatic clustering problem. Motivated by the application of managing traffic flows, Arkin et al. [8] studied a variety of chromatic 2-center clustering in 2D and gave both exact and approximate solutions. In addition, Ding and Xu [23, 25] studied chromatic clustering in high dimension; however, their method assumes that k is constant and thus it is unable to be extended to our general geometric prototype problem.

Computing the geometric matching $\mathcal{M}(A, B)$ is a sub-problem of geometric prototype. Besides the Hungarian algorithm [16], the computational geometry community has extensively studied its approximation algorithms for the case in constant dimension [2, 4, 7, 40, 41], and some of them can achieve nearly linear running time.

The rest of the paper is organized as follows. First, we introduce some basic results and useful tools in Section 2. Then we show our core-set construction and analysis in Section 3. Finally, we implement our algorithm and test it on multiple datasets in Section 4. Due to the space limit, we omit some proofs and the reader can find more details in the full version of our paper [21].

2 Preliminaries

The hardness. Actually, we are able to show that finding the optimal geometric prototype of a given instance is NP-hard and has no FPTAS even if $k = 2$ in high dimensional space, unless P=NP. Our proof makes use of the construction by Dasgupta for the NP-hardness proof of the 2-means clustering problem in high dimension [18].

The following lemma, which can be easily obtained via Definition 1, is repeatedly used in our analysis.

▶ **Lemma 3.** *Given three k-point sets A, B, and C in $\mathbb{R}^d$,*

$$\mathcal{M}(A, B) \leq 2\mathcal{M}(A, C) + 2\mathcal{M}(C, B). \tag{3}$$

Using Markov inequality and Lemma 3, Ding et al. [22] showed that a constant approximation can be achieved with constant probability.

▶ **Theorem 4** ([22]). *Let $\alpha > 1$. Given an instance $\mathbb{P}$ of the geometric prototype problem, if we randomly pick a point set P_{i_0} from $\mathbb{P}$, then with probability at least $1 - \frac{1}{\alpha}$, $\mathcal{M}(P_{i_0}, g(\mathbb{P}))$ is no larger than $\frac{\alpha}{n}\sum_{i=1}^{n}\mathcal{M}(P_i, g(\mathbb{P}))$ and P_{i_0} yields a $(2\alpha + 2)$-approximation.*

▶ Remark. To boost the success probability, we can try multiple times and select the one yielding the lowest objective value. For example, if we try t times, the success probability will be $1 - \frac{1}{\alpha^t}$.

According to Theorem 4, the selected P_{i_0} could serve as a good initialization for the geometric prototype. To further improve the approximation ratio, the algorithm in [22] adopts a simple alternating minimization procedure, i.e., alternatively updating the prototype and matchings round by round. The main drawback of this algorithm is that it needs to repeatedly compute the matchings between the prototype and all the given point sets in each round, and thus the running time is high especially when some or all of n, k, and d are large (as discussed at the beginning of Section 1.1).

In addition, we are able to apply the well known Johnson-Lindenstrauss (JL) lemma [1] to reduce the dimensionality before running the algorithm; also, the obtained geometric prototype in the lower dimension can be efficiently mapped back to the original space [22].

▶ **Theorem 5** ([22]). *Let $0 < \epsilon < 1$ and $c \geq 1$. Suppose we randomly project a given instance $\mathbb{P}$ of the geometric prototype problem from $\mathbb{R}^d$ to $\mathbb{R}^{O(\log(nk)/\epsilon^2)}$ and obtain a new instance $\mathbb{P}'$ in the lower dimension. Then, with high probability, we can convert any c-approximation for $\mathbb{P}'$ to a $c(\frac{1+\epsilon}{1-\epsilon})^2$-approximation for $\mathbb{P}$ in $\mathbb{R}^d$, in $O(nkd)$ time.*

The following lemma is a key tool in our analysis. In fact, it can be viewed as an interesting supplement of Lemma 3.

▶ **Lemma 6.** *Let A, B, and C be three k-point sets in $\mathbb{R}^d$. Then for any $\epsilon > 0$,*

$$\left|\mathcal{M}(A,B) - \mathcal{M}(A,C)\right| \leq (1 + \frac{1}{\epsilon})\mathcal{M}(B,C) + \epsilon\mathcal{M}(A,B) \qquad (4)$$

3 Core-set for Reducing the Data Size

Langberg and Schulman [32] introduced a framework of core-set (it was called "ϵ-approximator" in their paper) to compress data for several geometric shape fitting problems; further, Feldman and Langberg [26] improved the core-set size for a large class of clustering problems. Here, we consider constructing a core-set of the instance $\mathbb{P}$ so as to reduce the data size and running time. Formally, our objective is to find a small sample $\mathbb{S} \subset \mathbb{P}$ and assign a weight w_l for each $P_l \in \mathbb{S}$, such that for any k-point set $Q \subset \mathbb{R}^d$,

$$\left|\sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q) - \sum_{P_l \in \mathbb{S}} w_l \mathcal{M}(P_l, Q)\right| \leq O(\epsilon) \sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q) \qquad (5)$$

with certain probability and small enough $\epsilon > 0$. Moreover, we want to keep each weight w_l to be non-negative so as to easily run any existing algorithm or heuristic on the core-set.

Unfortunately, we cannot directly apply the existing ideas to the problem of geometric prototype, because the points from $\cup_{i=1}^n P_i$ are not independent from each other (due to the matching constraint in Definition 1; also see our remark below Definition 2) and it would be much more challenging to build the connection between the sampled core-set and $\mathbb{P}$. Instead, we regard each P_i as an "abstract point" and compute a core-set on these n abstract points. Though these abstract points can form some metric space with the matching costs being their pairwise (squared) distances, it is still quite different to metric clustering studied by [14, 26, 32], since the prototype $g(\mathbb{P})$ is not necessarily from $\mathbb{P}$ and could appear anywhere in the Euclidean space.

Conceptually, the core-set construction is a random sampling process: first, compute an upper bound on the sensitivity $\sigma_{\mathbb{P}}(P_i)$ of each P_i (we will formally define the sensitivity later); then take a sample from $\mathbb{P}$ with probabilities proportional to $\sigma_{\mathbb{P}}(P_i)$ to form the core-set. To implement this construction, we have to develop new ideas for resolving the following two issues. **(I)** How to compute $\sigma_{\mathbb{P}}(P_i)$, or its upper bound, so as to generate the probability distribution for sampling. **(II)** What sample size is needed to ensure our core-set yields a sufficient approximation. We consider these two issues in Section 3.1 and 3.2, respectively.

The final result for core-set construction of geometric prototype is presented in Theorem 13. We also discuss some extensions on other metrics (e.g., l_1 norm and earth mover's distance) and the time complexity in Section 3.3 and 3.4, respectively.

3.1 Solving Issue I

Following [32], the sensitivity of each $P_i \in \mathbb{P}$ is defined as follows:

$$\sigma_{\mathbb{P}}(P_i) = sup_Q \frac{\mathcal{M}(P_i, Q)}{\sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q)} \tag{6}$$

where Q is restricted to be k-point set in $\mathbb{R}^d$. Intuitively, the sensitivity measures the importance of each P_i among all the patterns of $\mathbb{P}$. Directly obtaining the value of $\sigma_{\mathbb{P}}(P_i)$ could be challenging and also needless, thus we often turn to compute an upper bound for it.

Recall that $g(\mathbb{P})$ is the optimal geometric prototype of $\mathbb{P}$, and let $\Delta = \sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, g(\mathbb{P}))$ for convenience.

▶ **Lemma 7.** *For any $P_i \in \mathbb{P}$, $\sigma_{\mathbb{P}}(P_i) \leq \frac{2\mathcal{M}(P_i, g(\mathbb{P}))}{\Delta} + \frac{16}{n}$.*

Proof. First, we consider $\frac{\mathcal{M}(P_i,Q)}{\sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l,Q)}$ with a fixed Q in (6). Through Lemma 3, we know that the numerator $\mathcal{M}(P_i, Q)$ is bounded by $2\mathcal{M}(P_i, g(\mathbb{P})) + 2\mathcal{M}(g(\mathbb{P}), Q)$. Then, we consider two cases: **(1)** $\mathcal{M}(g(\mathbb{P}), Q) \leq \frac{8}{n}\Delta$ and **(2)** $\mathcal{M}(g(\mathbb{P}), Q) > \frac{8}{n}\Delta$.

Since $\Delta \leq \sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q)$, we directly have

$$\begin{aligned} \frac{\mathcal{M}(P_i, Q)}{\sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q)} &\leq \frac{2\mathcal{M}(P_i, g(\mathbb{P})) + 2\mathcal{M}(g(\mathbb{P}), Q)}{\sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q)} \\ &\leq \frac{2\mathcal{M}(P_i, g(\mathbb{P})) + \frac{16}{n}\Delta}{\Delta} \\ &= \frac{2\mathcal{M}(P_i, g(\mathbb{P}))}{\Delta} + \frac{16}{n} \end{aligned} \tag{7}$$

for case (1).

Now, we assume that case (2) is true. Denote by $\mathbb{P}'$ the set $\{P_l \in \mathbb{P} \mid \mathcal{M}(P_l, g(\mathbb{P})) \leq \frac{2}{n}\Delta\}$, and Markov inequality implies $|\mathbb{P}'| \geq \frac{n}{2}$. Applying Lemma 3 again, we have

$$\begin{aligned} \sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q) &\geq \sum_{P_l \in \mathbb{P}'} \mathcal{M}(P_l, Q) \geq \sum_{P_l \in \mathbb{P}'} \Big(\frac{1}{2}\mathcal{M}(g(\mathbb{P}), Q) - \mathcal{M}(g(\mathbb{P}), P_l)\Big) \\ &\geq \sum_{P_l \in \mathbb{P}'} \Big(\frac{1}{2}\mathcal{M}(g(\mathbb{P}), Q) - \frac{2}{n}\Delta\Big) \geq \frac{n}{2}\Big(\frac{1}{2}\mathcal{M}(g(\mathbb{P}), Q) - \frac{2}{n}\Delta\Big) \\ &= \frac{n}{4}\mathcal{M}(g(\mathbb{P}), Q) - \Delta. \end{aligned} \tag{8}$$

As a consequence,

$$\frac{\mathcal{M}(P_i, Q)}{\sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q)} \leq \frac{2\mathcal{M}(P_i, g(\mathbb{P})) + 2\mathcal{M}(g(\mathbb{P}), Q)}{\frac{n}{4}\mathcal{M}(g(\mathbb{P}), Q) - \Delta}. \tag{9}$$

Since both $\mathcal{M}(P_i, g(\mathbb{P}))$ and Δ are independent of Q, the right-hand side of (9) can be viewed as a function on $\mathcal{M}(g(\mathbb{P}), Q)$. Through a simple calculation and the assumption of case (2) (i.e., $\mathcal{M}(g(\mathbb{P}), Q) > \frac{8}{n}\Delta$), we know that it is always less than $\frac{2\mathcal{M}(P_i, g(\mathbb{P}))}{\Delta} + \frac{16}{n}$.

Overall, we have $\sigma_{\mathbb{P}}(P_i) \leq \frac{2\mathcal{M}(P_i, g(\mathbb{P}))}{\Delta} + \frac{16}{n}$ for both cases. ◀

However, only Lemma 7 is not enough to compute the upper bound for $\sigma_{\mathbb{P}}(P_i)$, because neither $\mathcal{M}(P_i, g(\mathbb{P}))$ nor Δ is known. Therefore, we need to compute an approximation to replace the upper bound given by Lemma 7.

▶ **Lemma 8.** *Suppose P_{i_0} is randomly picked from $\mathbb{P}$, and let $\tilde{\Delta} = \sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, P_{i_0})$ and $\alpha > 1$. Then with probability $1 - \frac{1}{\alpha}$, for all $1 \leq i \leq n$, $\sigma_{\mathbb{P}}(P_i) \leq 8(\alpha+1)\frac{\mathcal{M}(P_i, P_{i_0})}{\tilde{\Delta}} + \frac{4\alpha+16}{n}$.*

Proof. According to Theorem 4, we know that $\mathcal{M}(P_{i_0}, g(\mathbb{P})) \leq \frac{\alpha}{n}\Delta$ and $\tilde{\Delta} \leq 2(\alpha+1)\Delta$ with probability at least $1 - \frac{1}{\alpha}$. Then we have

$$\begin{aligned} \sigma_{\mathbb{P}}(P_i) &\leq \frac{2\mathcal{M}(P_i, g(\mathbb{P}))}{\Delta} + \frac{16}{n} \leq \frac{4\mathcal{M}(P_i, P_{i_0}) + 4\mathcal{M}(P_{i_0}, g(\mathbb{P}))}{\Delta} + \frac{16}{n} \\ &\leq \frac{4\mathcal{M}(P_i, P_{i_0})}{\frac{1}{2(\alpha+1)}\tilde{\Delta}} + \frac{4\mathcal{M}(P_{i_0}, g(\mathbb{P}))}{\Delta} + \frac{16}{n} \\ &\leq 8(\alpha+1)\frac{\mathcal{M}(P_i, P_{i_0})}{\tilde{\Delta}} + \frac{4\alpha+16}{n}, \end{aligned} \tag{10}$$

where the first inequality comes from Lemma 7. So the proof is completed. ◀

Lemma 8 indicates that once P_{i_0} is selected, we can obtain an upper bound for each $\sigma_{\mathbb{P}}(P_i)$ by computing the values $\mathcal{M}(P_i, P_{i_0})$ and $\tilde{\Delta}$.

3.2 Solving Issue II

Let $t_{\mathbb{P}}(P_i)$ and T denote the obtained upper bound of $\sigma_{\mathbb{P}}(P_i)$ from Lemma 8 and their sum, respectively. It is easy to know that $T = \sum_{P_i \in \mathbb{P}} t_{\mathbb{P}}(P_i) \leq 8(\alpha+1) + 4\alpha + 16$ which is constant if α is constant. For the sake of simplicity, we always assume $T = O(1)$ in our analysis below.

We have the following theorem from [32, 45] (we slightly modify their statements to fit our problem better).

▶ **Theorem 9** ([32, 45]). *Let Q be any fixed k-point set in $\mathbb{R}^d$. **i.** If we take a sample P_i from $\mathbb{P}$ according to the distribution $\frac{t_{\mathbb{P}}(P_i)}{T}$, the expectation of $\frac{T}{t_{\mathbb{P}}(P_i)}\mathcal{M}(P_i, Q)$ is $\sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q)$. **ii.** If we take a sample $\mathbb{S}$ of size of r from $\mathbb{P}$ according to the same distribution, and let $\epsilon > 0$,*

$$Pr\left[\left| \sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q) - \frac{1}{r}\sum_{P_l \in \mathbb{S}} \frac{T}{t_{\mathbb{P}}(P_l)}\mathcal{M}(P_l, Q)\right| \leq \epsilon \sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q)\right] \geq 1 - 2e^{-\frac{2r\epsilon^2}{T^2}}. \tag{11}$$

In particular, (11) is an application of Hoeffding's inequality because each $\frac{T}{t_{\mathbb{P}}(P_i)}\mathcal{M}(P_i, Q)$ is a random variable between 0 and $T\sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q)$ (see Lemma 2.2 of [45] for more details). Moreover, (11) shows that the sample $\mathbb{S}$ together with the weight $w_l = \frac{1}{r}\frac{T}{t_{\mathbb{P}}(P_l)}$ for each $P_l \in \mathbb{S}$ will form a core-set of $\mathbb{P}$ with respect to the fixed Q (see (5)). But (5) should hold for an infinite number of possible candidates for the geometric prototype, rather than one single Q, in the space. Hence, we need to determine an appropriate sample size (i.e., issue **(II)**).

Our basic idea is to discretize the space and generate a finite number of representations for them; then we can take a union bound for the final success probability through (11). Note [45] also used discretization to determine the sample size for projective clustering integer points; but our idea and analysis are quite different due to the different natures of the problems. Also, [26, 32] defined the "dimension" of the clustering problems so as to bound their sample sizes. Here, we avoid using their approach due to two reasons: first, it will be very complicated to define and compute the dimension of the geometric prototype problem; second, the framework in [26] would result in a more complicated sampling process and even may cause negative weights, however, we prefer to keep our sampling process simple as described in Theorem 9 (especially when using any available algorithm or heuristic as a black box on the core-set). We elaborate on our analysis below.

By Theorem 4, we assume that a randomly picked P_{i_0} yields a $(2\alpha+2)$-approximation and denote by L the resulting cost $\sum_{P_l\in\mathbb{P}}\mathcal{M}(P_l,P_{i_0})$. The following lemma reveals that we just need to consider the k-point sets which are not too far from P_{i_0}.

▶ **Lemma 10.** *For any k-point set Q with $\mathcal{M}(Q,P_{i_0})>\frac{4L}{n}$, the resulting cost $\sum_{P_l\in\mathbb{P}}\mathcal{M}(P_l,Q)$ is always higher than $\sum_{P_l\in\mathbb{P}}\mathcal{M}(P_l,P_{i_0})$.*

Proof. Using Lemma 3, we have

$$\sum_{P_l\in\mathbb{P}}\mathcal{M}(P_l,Q)\geq\sum_{P_l\in\mathbb{P}}\left(\frac{1}{2}\mathcal{M}(Q,P_{i_0})-\mathcal{M}(P_l,P_{i_0})\right)>\frac{1}{2}4L-L=\sum_{P_l\in\mathbb{P}}\mathcal{M}(P_l,P_{i_0}). \tag{12}$$

So the proof is completed. ◀

Because we already have the initial solution P_{i_0}, we are only interested in the solutions having lower costs. Thus, we focus on the k-point set Qs with $\mathcal{M}(Q,P_{i_0})\leq\frac{4L}{n}$ based on Lemma 10. Let $Q=\{q_1,q_2,\cdots,q_k\}\subset\mathbb{R}^d$ and $R=L/n$. W.l.o.g, we assume the induced permutation of $\mathcal{M}(Q,P_{i_0})$ in Definition 1 is $\pi(j)=j$ for $1\leq j\leq k$. The constraint $\mathcal{M}(Q,P_{i_0})\leq\frac{4L}{n}$ directly implies that $||q_j-p^{i_0}_j||\leq2\sqrt{R}$ for each $1\leq j\leq k$. We use $B(x,\rho)$ to denote the ball centered at the point x with the radius ρ. Then we draw k balls $B(p^{i_0}_j,2\sqrt{R})$ for each $1\leq j\leq k$; inside each ball, we build a uniform grid G_j with the grid side length $\epsilon\sqrt{\frac{R}{kd}}$. Let Γ be the Cartesian product $G_1\times G_2\times\cdots\times G_k$. It is easy to know that Γ contains $O\Big((\frac{4\sqrt{kd}}{\epsilon})^{kd}\Big)$ k-point sets in total. Therefore, we can apply (11) of Theorem 9 to obtain a union bound over all the k-point sets of Γ (recall $T=O(1)$).

▶ **Lemma 11.** *If the sample $\mathbb{S}$ in Theorem 9 has the size of $O(\frac{kd}{\epsilon^2}\log\frac{kd}{\epsilon})$, and each $P_l\in\mathbb{S}$ has the weight $w_l=\frac{1}{r}\frac{T}{t_{\mathbb{P}}(P_l)}$, then with constant probability the inequality (5) holds for each $Q\in\Gamma$.*

Next we consider the k-point set $Q=\{q_1,q_2,\cdots,q_k\}\notin\Gamma$. Again, w.l.o.g, we assume the induced permutation of $\mathcal{M}(P_{i_0},Q)$ is $\pi(j)=j$ for $1\leq j\leq k$. Also, due to our above assumption, we know that each q_j is covered by the ball $B(p^{i_0}_j,2\sqrt{R})$. To help our analysis, we take its "nearest neighbor" from Γ, $\mathcal{N}(Q)=\{\mathcal{N}(q_1),\mathcal{N}(q_2),\cdots,\mathcal{N}(q_k)\}$ with each $\mathcal{N}(q_j)$ being the nearest grid point of q_j in G_j. So we have

$$||q_j-\mathcal{N}(q_j)||\leq\epsilon\sqrt{\frac{R}{k}}\quad for\quad 1\leq j\leq k. \tag{13}$$

For the sake of convenience, let $X_1=\Big|\sum_{P_l\in\mathbb{P}}\mathcal{M}(P_l,Q)-\sum_{P_l\in\mathbb{P}}\mathcal{M}(P_l,\mathcal{N}(Q))\Big|$, $X_2=\Big|\sum_{P_l\in\mathbb{P}}\mathcal{M}(P_l,\mathcal{N}(Q))-\frac{1}{r}\sum_{P_l\in\mathbb{S}}\frac{T}{t_{\mathbb{P}}(P_l)}\mathcal{M}(P_l,\mathcal{N}(Q))\Big|$, $X_3=\Big|\frac{1}{r}\sum_{P_l\in\mathbb{S}}\frac{T}{t_{\mathbb{P}}(P_l)}\mathcal{M}(P_l,\mathcal{N}(Q))-\frac{1}{r}\sum_{P_l\in\mathbb{S}}\frac{T}{t_{\mathbb{P}}(P_l)}\mathcal{M}(P_l,Q)\Big|$. It is easy to see

$$\Big|\sum_{P_l\in\mathbb{P}}\mathcal{M}(P_l,Q)-\frac{1}{r}\sum_{P_l\in\mathbb{S}}\frac{T}{t_{\mathbb{P}}(P_l)}\mathcal{M}(P_l,Q)\Big|\leq X_1+X_2+X_3 \tag{14}$$

where X_2 is bounded by Lemma 11. So the remaining issue is to prove that the other two items X_1 and X_3 in (14) are small as well. That is, Lemma 11 can be extended from $\mathcal{N}(Q)$ to Q.

Note $X_1 \leq \sum_{P_l \in \mathbb{P}} \left|\mathcal{M}(P_l, Q) - \mathcal{M}(P_l, \mathcal{N}(Q))\right|$, so we consider each item $\left|\mathcal{M}(P_l, Q) - \mathcal{M}(P_l, \mathcal{N}(Q))\right|$ separately. Using Lemma 6, we have

$$\left|\mathcal{M}(P_l, Q) - \mathcal{M}(P_l, \mathcal{N}(Q))\right| \leq (1 + \frac{1}{\epsilon})\mathcal{M}(Q, \mathcal{N}(Q)) + \epsilon\mathcal{M}(P_l, Q). \tag{15}$$

In addition, we have $\mathcal{M}(Q, \mathcal{N}(Q)) \leq k\left(\epsilon\sqrt{\frac{R}{k}}\right)^2 = \epsilon^2 R$ by (13). Therefore, we have

$$\begin{aligned}\left|\sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q) - \sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, \mathcal{N}(Q))\right| &\leq \sum_{P_l \in \mathbb{P}} \left|\mathcal{M}(P_l, Q) - \mathcal{M}(P_l, \mathcal{N}(Q))\right| \\ &\leq (1 + \frac{1}{\epsilon})n\mathcal{M}(Q, \mathcal{N}(Q)) + \epsilon \sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q) \\ &\leq O(\epsilon)nR + \epsilon \sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q) \\ &= O(\epsilon) \sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q), \end{aligned} \tag{16}$$

where the last equality comes from $nR = L$ which is a constant approximation of the optimal objective value. (16) also implies that

$$\left(1 - O(\epsilon)\right) \sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q) \leq \sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, \mathcal{N}(Q)) \leq \left(1 + O(\epsilon)\right) \sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, Q). \tag{17}$$

Next, we consider the last item X_3 in (14). It is a little more complicated because the coefficient $\frac{T}{t_{\mathbb{P}}(P_l)}$ could be large. We need the following lemma first.

▶ **Lemma 12.** *For each $P_l \in \mathbb{P}$, $t_{\mathbb{P}}(P_l) > \frac{1}{4n}$.*

Proof. Fix one $P_l \in \mathbb{P}$. We select $P_{l'}$ that has the largest matching cost to P_l, i.e., $\mathcal{M}(P_l, P_{l'}) = \max_{P_i \in \mathbb{P}} \mathcal{M}(P_l, P_i)$, and set $Q = P_{l'}$. Using Lemma 3, we have $\mathcal{M}(P_i, Q) \leq 2\mathcal{M}(P_i, P_l) + 2\mathcal{M}(P_l, Q) \leq 4\mathcal{M}(P_l, Q)$ for any $1 \leq i \leq n$. Therefore, based on the fact that $t_{\mathbb{P}}(P_l)$ is the upper bound of $\sigma_{\mathbb{P}}(P_l)$ in (6), we know that it should be at least $\frac{\mathcal{M}(P_l, Q)}{(1+4(n-1))\mathcal{M}(P_l, Q)} > \frac{1}{4n}$. ◀

Using Lemma 12 and the same idea for (16), we have

$$\begin{aligned} &\left|\frac{1}{r}\sum_{P_l \in \mathbb{S}} \frac{T}{t_{\mathbb{P}}(P_l)}\mathcal{M}(P_l, \mathcal{N}(Q)) - \frac{1}{r}\sum_{P_l \in \mathbb{S}} \frac{T}{t_{\mathbb{P}}(P_l)}\mathcal{M}(P_l, Q)\right| \\ \leq\ & \frac{1}{r}\sum_{P_l \in \mathbb{S}} \frac{T}{t_{\mathbb{P}}(P_l)}\left|\mathcal{M}(P_l, \mathcal{N}(Q)) - \mathcal{M}(P_l, Q)\right| \\ \leq\ & \frac{1}{r}\sum_{P_l \in \mathbb{S}} \frac{T}{t_{\mathbb{P}}(P_l)}\Big((1 + \frac{1}{\epsilon})\mathcal{M}(\mathcal{N}(Q), Q) + \epsilon\mathcal{M}(P_l, \mathcal{N}(Q))\Big) \\ \leq\ & \max_{P_l \in \mathbb{S}}\{\frac{T}{t_{\mathbb{P}}(P_l)}\} \cdot (1 + \frac{1}{\epsilon})\mathcal{M}(\mathcal{N}(Q), Q) + \epsilon\frac{1}{r}\sum_{P_l \in \mathbb{S}} \frac{T}{t_{\mathbb{P}}(P_l)}\mathcal{M}(P_l, \mathcal{N}(Q)) \\ \leq\ & O(\epsilon)nR + \epsilon\frac{1}{r}\sum_{P_l \in \mathbb{S}} \frac{T}{t_{\mathbb{P}}(P_l)}\mathcal{M}(P_l, \mathcal{N}(Q)), \end{aligned} \tag{18}$$

where the last inequality comes from Lemma 12 and $T = O(1)$. In addition, Lemma 11 guarantees that $\epsilon\frac{1}{r}\sum_{P_l \in \mathbb{S}} \frac{T}{t_{\mathbb{P}}(P_l)}\mathcal{M}(P_l, \mathcal{N}(Q)) = O(\epsilon)\sum_{P_l \in \mathbb{P}} \mathcal{M}(P_l, \mathcal{N}(Q))$. Applying the triangle inequality (14) with the bounds (16), (17) and (18), we have

$$\begin{aligned}
&\Big|\sum_{P_l\in\mathbb{P}}\mathcal{M}(P_l,Q)-\frac{1}{r}\sum_{P_l\in\mathbb{S}}\frac{T}{t_{\mathbb{P}}(P_l)}\mathcal{M}(P_l,Q)\Big| \\
\leq\;& O(\epsilon)\sum_{P_l\in\mathbb{P}}\mathcal{M}(P_l,Q)+O(\epsilon)\sum_{P_l\in\mathbb{P}}\mathcal{M}(P_l,\mathcal{N}(Q)) \\
=\;& O(\epsilon)\sum_{P_l\in\mathbb{P}}\mathcal{M}(P_l,Q). \qquad (19)
\end{aligned}$$

Consequently, we have the final theorem for core-set.

▶ **Theorem 13.** *Let P_{i_0} be the k-point set randomly selected by Theorem 4, and $\mathbb{S}$ be the sample from $\mathbb{P}$ according to the distribution $\frac{t_{\mathbb{P}}(P_i)}{T}$. If the sample $\mathbb{S}$ has the size of $r=O(\frac{kd}{\epsilon^2}\log\frac{kd}{\epsilon})$ and each $P_l\in\mathbb{S}$ has the weight $w_l=\frac{1}{r}\frac{T}{t_{\mathbb{P}}(P_l)}$, then with constant probability the inequality (5) holds for any k-point set $Q\subset\mathbb{R}^d$ with $\mathcal{M}(Q,P_{i_0})\leq\frac{4L}{n}$.*

Recall that Theorem 5 tells us that the dimension can be reduced by Johnson-Lindenstrauss (JL)-transform. Thus, we directly have the following corollary.

▶ **Corollary 14.** *Given a high dimensional instance $\mathbb{P}$ (e.g., $d\gg\log n,\log k$), we can obtain a sample $\mathbb{S}$ having the size of $\tilde{O}(\frac{k}{\epsilon^4})$, where with constant probability the inequality (5) holds for any k-point set $Q\subset\mathbb{R}^d$ with $\mathcal{M}(Q,P_{i_0})\leq\frac{4L}{n}$. $\tilde{O}(\cdot)$ ignores the logarithmic factors $\log n$ and $\log k$.*

3.3 Some Extensions

Here, we briefly introduce some extensions of our core-set construction on other metrics.
(1). Our core-set construction can be extended to l_1 norm, i.e., the squared distances are replaced by absolute distances in the matching cost (1). Actually, the analysis for l_1 norm is even easier than that for l_2 norm, since we can directly use the triangle inequality rather than Lemma 3 or Lemma 6 when solving the aforementioned two issues, bounding the sensitivities and discretizing the space of candidates for geometric prototype.

A remaining issue for future work is that the dimension reduction result of Theorem 5 is not applicable to l_1 norm, due to the fact that it is much harder to compute geometric median (Fermat-Weber point) than mean point [15]. Fortunately, the high dimensional application, ensemble clustering, mentioned in Section 1 only uses l_2 norm, because the symmetric difference between two clusters corresponds to their squared distance in the space.
(2). We can also consider the case with weighted point sets for both l_1 and l_2 norm, i.e., each point of P_i has a non-negative weight. To make the problem meaningful in practice, we require that each P_i and the desired geometric prototype have the same total weight $W>0$; we can further assume W and all the weights are integers by scaling and rounding in practice. Thus, the computation on the matching between two point sets becomes the problem of earth mover's distance (EMD) [39]. Fortunately, the triangle inequality still holds for EMD because we assume they have equal total weight; as a consequence, we can bound the sensitivities for issue **(I)**. For issue **(II)**, we still discretize the space and build the set of k-point sets Γ with the same cardinality of the unweighted case; the only difference is that we need to consider the total $O(W^k)$ possible distributions of the total weight W over the k points of each k-point set, which increases the size of the core-set with an extra $O(\frac{k\log W}{\epsilon^2})$.

3.4 The Time Complexity

Suppose the complexity of computing $\mathcal{M}(A, B)$ is $h(k, d)$, then the running time for computing the core-set is simply $O(h(k, d) \cdot n)$ because we just need to compute each $\mathcal{M}(P_i, P_{i_0})$ so as to obtain the sensitivities for sampling (see Lemma 8). For simplicity, we can just use the Hungarian algorithm [16] so that $h(k, d) = O(k^2 d + k^3)$, where the term $k^2 d$ is for building the bipartite graph. In fact, this can be further improved by our following two observations. First, we just need to know the matching costs, rather than the matchings, for computing the upper bounds of the sensitivities in Lemma 8. Second, it is not necessary to always have the optimal matching costs. For example, if we compute a value $\mathcal{M}'(P_i, P_{i_0})$ for each $\mathcal{M}(P_i, P_{i_0})$ instead, such that $\mathcal{M}(P_i, P_{i_0}) \leq \mathcal{M}'(P_i, P_{i_0}) \leq c\mathcal{M}(P_i, P_{i_0})$ with some constant $c \geq 1$, the resulting T and each $t_{\mathbb{P}}(P_i)$ will increase by some appropriate constant factors correspondingly; in other words, the sample size in Theorem 13 will increase by only a constant factor. Some algorithms [13, 30, 31] are designed for approximately estimating the matching cost, and their running times can be nearly linear if the dimension d is constant; in practical fields, several heuristic algorithms [37] are also proposed for this purpose.

For the high dimensional case, we can apply JL-transform in advance, to reduce the dimensionality to be $O(\log(nk)/\epsilon^2)$ (Theorem 5 and Corollary 14). A naive implementation of the JL-transform by matrix multiplication has the complexity $O\big(\frac{1}{\epsilon^2} nkd \log(nk)\big)$ [19], and several even faster and practical algorithms have been studied before [1, 6, 34].

4 Experiments

To show the advantage of using core-sets for the problem of geometric prototype, we study the two important applications introduced in Section 1, Wasserstein barycenter and ensemble clustering. For each application, we run the existing algorithm on the original dataset and core-sets with different size levels. In general, our experiments suggest that running the algorithm on a small core-set can achieve very close performance and greatly reduce the running time. All of the experimental results were obtained on a Windows workstation with 2.4GHz Intel Xeon E5-2630 v3 CPU and 32GB DDR4 2133MHz Memory; the algorithms are implemented in Matlab R2016b.

Wasserstein barycenter. MNIST [33] is a popular benchmark dataset of handwritten digits from 0 to 9. For each digit, we generate a set of 3000 28×28 grayscale images including 10% noise (i.e., 300 images randomly selected from the other 9 digits). First, we represent the 28×28 pixels by 60 weighted $2D$ points via k-means clustering [36]: group the pixels into 60 clusters and each cluster is represented by its cluster center; each center has the weight equal to the total pixel values of the cluster. Therefore the problem of Wasserstein barycenter becomes an instance of geometric prototype with $n = 3000$, $k = 60$, and $d = 2$.

Ensemble clustering. To construct an instance of ensemble clustering, we generate a synthetic dataset of 2000 points randomly sampled from $k = 50$ Gaussian distributions in $\mathbb{R}^{100}$; we apply k-means clustering 1000 times, where each time has a different initialization for the k mean points, to generate 1000 different clustering solutions. According to the model introduced by [22], each instance is a geometric prototype problem with 1000 different 50-point sets in $\mathbb{R}^{2000}$. We apply JL-transform to reduce the dimensionality from 2000 to 100, before constructing the core-set and running the algorithm; we just use the simplest random matrix multiplication to implement JL-transform [19] (actually this step takes about only 5% of the whole running time in the experiments).

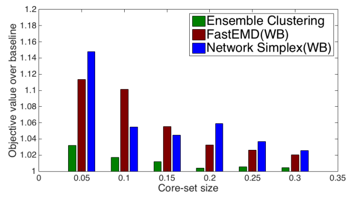


Figure 2 Normalized objective value.

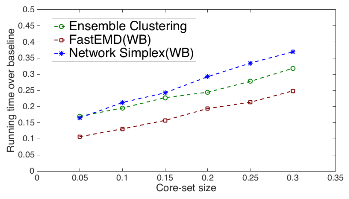


Figure 3 Normalized running time.

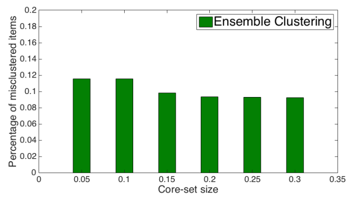


Figure 4 Percentage of misclustered items.

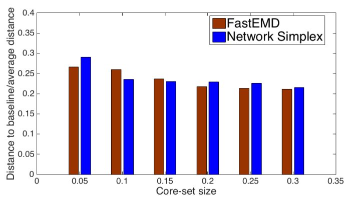


Figure 5 Matching cost to ground truth.

For both applications, we construct the core-sets using the method in Section 3; we vary the core-set size from 5% to 30% of the input size. To construct the core-set, we need to compute the matching cost $\mathcal{M}(P_i, P_{i_0})$ as discussed in Section 3.4: for the high dimensional application (i.e., ensemble clustering), we just use the Hungarian algorithm [16]; for the low dimensional application (i.e., Wasserstein barycenter), we use two existing popular algorithms for computing EMD, *Network simplex algorithm* [5] and the heuristic but faster algorithm *FastEMD* [37]. As the black box for computing the geometric prototype, we use the alternating minimization approach [22]. For each application, we consider three criteria: running time, objective value (in Definition 2), and difference to ground truth. For ensemble clustering, we compute the percentage of misclustered items of the obtained prototype as the difference to ground truth. For Wasserstein barycenter, since it is difficult to determine a unique ground truth for each handwritten digit, we directly use the prototype obtained from the original input dataset as the ground truth; then we compute its matching cost to the prototype obtained from core-set, denoted by x, as well as the average matching cost over the input images to the ground truth, denoted by Ave; finally, we obtain the ratio x/Ave. In general, the lower the ratio x/Ave, the closer the obtained prototype to the ground truth (comparing with the input images).

Results. For each application, we run 50 trials and report the average results. Figure 2 shows the obtained normalized objective values over the base line (i.e., the objective value obtained on the original input dataset), which are all lower than 1.2; that means our core-sets are good approximations for the original data. More importantly, the running times are significantly reduced in Figure 3, e.g., for the core-set having 5% of the input data size, the algorithm (containing the core-sets construction) only runs within 10%-17% of the original time. In addition, our obtained prototypes are very close to the corresponding ground truths, even for the core-set at the level 5%. Figure 4 provides the percentages of misclustered items for ensemble clustering, which are around 8%-12%. Figure 5 shows the values of x/Ave, which are around 0.25. For Wasserstein barycenter, we can see the Network simplex algorithm and FastEMD algorithm achieve very similar qualities, but FastEMD only takes about 60% of the running time of the Network simplex algorithm.

5 Future Work

Following our work, several interesting problems for geometric prototype deserve to be explored. For example, is there any algorithm achieving a better approximation ratio than Theorem 4? In addition, we leave the hardness for the low dimensional case of geometric prototype as an open problem in future work.

References

1 Dimitris Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

2 Pankaj K. Agarwal, Kyle Fox, Debmalya Panigrahi, Kasturi R. Varadarajan, and Allen Xiao. Faster algorithms for the geometric transportation problem. In *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*, pages 7:1–7:16, 2017.

3 Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Geometric approximation via coresets. *Combinatorial and Computational Geometry*, 52:1–30, 2005.

4 Pankaj K. Agarwal and Kasturi R. Varadarajan. A near-linear constant-factor approximation for euclidean bipartite matching? In *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*, pages 247–252, 2004.

5 Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network flows: theory, algorithms, and applications.* Prentice Hall, 1993.

6 Nir Ailon and Bernard Chazelle. The fast Johnson—Lindenstrauss transform and approximate nearest neighbors. *SIAM Journal on computing*, 39(1):302–322, 2009.

7 Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 574–583, 2014.

8 Esther M Arkin, José Miguel Díaz-Báñez, Ferran Hurtado, Piyush Kumar, Joseph S.B. Mitchell, Belén Palop, Pablo Pérez-Lantero, Maria Saumell, and Rodrigo I Silveira. Bichromatic 2-center of pairs of points. *Computational Geometry*, 48(2):94–107, 2015.

9 Marcus Baum, Peter Willett, and Uwe D. Hanebeck. On wasserstein barycenters and MMOSPA estimation. *IEEE Signal Process. Lett.*, 22(10):1511–1515, 2015.

10 Jean-David Benamou, Guillaume Carlier, Marco Cuturi, Luca Nenna, and Gabriel Peyré. Iterative bregman projections for regularized transportation problems. *SIAM Journal on Scientific Computing*, 37(2):A1111–A1138, 2015.

11 Paola Bonizzoni, Gianluca Della Vedova, Riccardo Dondi, and Tao Jiang. On the approximation of correlation clustering and consensus clustering. *Journal of Computer and System Sciences*, 74(5):671–696, 2008.

12 Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

13 Sergio Cabello, Panos Giannopoulos, Christian Knauer, and Günter Rote. Matching point sets with respect to the earth mover's distance. *Computational Geometry*, 39(2):118–133, 2008.

14 Ke Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009.

15 Michael B Cohen, Yin Tat Lee, Gary Miller, Jakub Pachocki, and Aaron Sidford. Geometric median in nearly linear time. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 9–21. ACM, 2016.

16 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

17 Marco Cuturi and Arnaud Doucet. Fast computation of wasserstein barycenters. In *International Conference on Machine Learning*, pages 685–693, 2014.

18 Sanjoy Dasgupta. The hardness of k-means clustering. *Technical Report*, 2008.

19 Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003.

20 Hu Ding, Ronald Berezney, and Jinhui Xu. k-prototype learning for 3d rigid structures. In *Advances in Neural Information Processing Systems*, pages 2589–2597, 2013.

21 Hu Ding and Manni Liu. On geometric prototype and applications. *CoRR*, abs/1804.09655, 2018. `arXiv:1804.09655`.

22 Hu Ding, Lu Su, and Jinhui Xu. Towards distributed ensemble clustering for networked sensing systems: a novel geometric approach. In *Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc 2016, Paderborn, Germany, July 4-8, 2016*, pages 1–10, 2016.

23 Hu Ding and Jinhui Xu. Solving the chromatic cone clustering problem via minimum spanning sphere. In *Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 773–784, 2011.

24 Hu Ding and Jinhui Xu. Finding median point-set using earth mover's distance. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

25 Hu Ding and Jinhui Xu. A unified framework for clustering constrained data without locality property. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1471–1490, 2015.

26 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 569–578, 2011.

27 Jing Gao, Feng Liang, Wei Fan, Yizhou Sun, and Jiawei Han. Graph-based consensus maximization among multiple supervised and unsupervised models. In *Advances in Neural Information Processing Systems*, pages 585–593, 2009.

28 Joydeep Ghosh and Ayan Acharya. Cluster ensembles: Theory and applications. In *Data Clustering: Algorithms and Applications*, pages 551–570. CRC, 2013.

29 Alexandre Gramfort, Gabriel Peyré, and Marco Cuturi. Fast optimal transport averaging of neuroimaging data. In *International Conference on Information Processing in Medical Imaging*, pages 261–272. Springer, 2015.

30 Piotr Indyk. A near linear time constant factor approximation for euclidean bichromatic matching (cost). In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 39–42. Society for Industrial and Applied Mathematics, 2007.

31 Piotr Indyk and Nitin Thaper. Fast color image retrieval via embeddings. In *Workshop on Statistical and Computational Theories of Vision (at ICCV)*, 2003.

32 Michael Langberg and Leonard J Schulman. Universal ε-approximators for integrals. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 598–607. SIAM, 2010.

33 Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

34 Edo Liberty and Steven W Zucker. The mailman algorithm: A note on matrix–vector multiplication. *Information Processing Letters*, 109(3):179–182, 2009.

35 Jialu Liu, Chi Wang, Jing Gao, and Jiawei Han. Multi-view clustering via joint nonnegative matrix factorization. In *Proc. of SDM*, volume 13, pages 252–260, 2013.

36 Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

37 Ofir Pele and Michael Werman. Fast and robust earth mover's distances. In *Computer vision, 2009 IEEE 12th international conference on*, pages 460–467. IEEE, 2009.

38 Jeff M. Phillips. Coresets and sketches. *Computing Research Repository*, 2016.

39 Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

40 R. Sharathkumar and Pankaj K. Agarwal. Algorithms for the transportation problem in geometric settings. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 306–317, 2012.

41 R. Sharathkumar and Pankaj K. Agarwal. A near-linear time ϵ-approximation algorithm for geometric bipartite matching. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 385–394, 2012.

42 Vikas Singh, Lopamudra Mukherjee, Jiming Peng, and Jinhui Xu. Ensemble clustering using semidefinite programming with applications. *Machine learning*, 79(1-2):177–200, 2010.

43 Matthew Staib, Sebastian Claici, Justin Solomon, and Stefanie Jegelka. Parallel streaming wasserstein barycenters. *arXiv preprint arXiv:1705.07443*, 2017.

44 Alexander Strehl and Joydeep Ghosh. Cluster ensembles-a knowledge reuse framework for combining partitionings. In *AAAI/IAAI*, pages 93–99, 2002.

45 Kasturi R. Varadarajan and Xin Xiao. A near-linear algorithm for projective clustering integer points. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1329–1342, 2012.

46 Jianbo Ye, Panruo Wu, James Z Wang, and Jia Li. Fast discrete distribution clustering using wasserstein barycenter with sparse support. *IEEE Transactions on Signal Processing*, 65(9):2317–2332, 2017.

Improved Bounds for Multipass Pairing Heaps and Path-Balanced Binary Search Trees

Dani Dorfman
Blavatnik School of Computer Science, Tel Aviv University, Israel
dannatand@mail.tau.ac.il

Haim Kaplan[1]
Blavatnik School of Computer Science, Tel Aviv University, Israel
haimk@post.tau.ac.il

László Kozma[2]
Eindhoven University of Technology, The Netherlands
lkozma@gmail.com

Seth Pettie[3]
University of Michigan
pettie@umich.edu

Uri Zwick[4]
Blavatnik School of Computer Science, Tel Aviv University, Israel
zwick@tau.ac.il

Abstract

We revisit *multipass* pairing heaps and *path-balanced* binary search trees (BSTs), two classical algorithms for data structure maintenance. The pairing heap is a simple and efficient "self-adjusting" heap, introduced in 1986 by Fredman, Sedgewick, Sleator, and Tarjan. In the multipass variant (one of the original pairing heap variants described by Fredman et al.) the minimum item is extracted via repeated *pairing rounds* in which neighboring siblings are linked.

Path-balanced BSTs, proposed by Sleator (cf. Subramanian, 1996), are a natural alternative to Splay trees (Sleator and Tarjan, 1983). In a path-balanced BST, whenever an item is accessed, the search path leading to that item is re-arranged into a balanced tree.

Despite their simplicity, both algorithms turned out to be difficult to analyse. Fredman et al. showed that operations in multipass pairing heaps take amortized $O(\log n \cdot \log\log n/\log\log\log n)$ time. For searching in path-balanced BSTs, Balasubramanian and Raman showed in 1995 the same amortized time bound of $O(\log n \cdot \log\log n/\log\log\log n)$, using a different argument.

In this paper we show an explicit connection between the two algorithms and improve both bounds to $O\left(\log n \cdot 2^{\log^* n} \cdot \log^* n\right)$, respectively $O\left(\log n \cdot 2^{\log^* n} \cdot (\log^* n)^2\right)$, where $\log^*(\cdot)$ denotes the slowly growing iterated logarithm function. These are the first improvements in more than three, resp. two decades, approaching the information-theoretic lower bound of $\Omega(\log n)$.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases data structure, priority queue, pairing heap, binary search tree

[1] Research supported by The Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11), Israel Science Foundation grant no. 1841-14.
[2] Supported by ERC grant no. 617951. Work done while at Tel Aviv University, research supported by The Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).
[3] Supported by NSF grants CCF-1514383 and CCF-1637546.
[4] Research supported by BSF grant no. 2012338 and by The Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).

© Dani Dorfman, Haim Kaplan, László Kozma, Seth Pettie, and Uri Zwick;
licensed under Creative Commons License CC-BY

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.24

Related Version A full version of the paper is available at `https://arxiv.org/abs/1806.08692`.

1 Introduction

Binary search trees (BSTs) and heaps are the canonical comparison-based implementations of the well-known *dictionary* and *priority queue* data types.

In a balanced **BST** all standard dictionary operations (*insert*, *delete*, *search*) take $O(\log n)$ time, where n is the size of the dictionary. Early research has mostly focused on structures that are kept (approximately) balanced throughout their usage. (AVL-, red-black-trees, and randomized treaps are important examples, see e.g., [11, § 6.2.2]). These data structures re-balance themselves when necessary, guided by auxiliary data stored in every node.

By contrast, Splay trees (Sleator, Tarjan, 1983 [17]) achieve $O(\log n)$ amortized time per operation without any explicit balancing strategy and with no bookkeeping whatsoever. Instead, Splay trees re-adjust the search path *after every access*, in a way that depends only on the shape of the search path, ignoring the global structure of the tree. Besides the $O(\log n)$ amortized time, Splay trees are known to satisfy stronger, adaptive properties (see [9, 3] for surveys). They are, in fact, conjectured to be optimal on every sequence of operations (up to a constant factor); this is the famous "dynamic optimality conjecture" [17]. Splay trees and data structures of a similar flavor (i.e., local restructuring, adaptivity, no auxiliary data) are called "self-adjusting".

The efficiency of Splay trees is intriguing and counter-intuitive. They re-arrange the search path by a sequence of double rotations ("zig-zig" and "zig-zag"), bringing the accessed item to the root. It is not hard to see that this transformation results in "approximate depth-halving" for the nodes on the search path; the connection between this depth-halving and the overall efficiency of Splay trees is, however, far from obvious.

An arguably more natural approach for BST re-adjustment would be to turn the search path, after every search, into a balanced tree.[5] This strategy combines the idea of self-adjusting trees with the more familiar idea of balancedness. Indeed, this algorithm was proposed early on by Sleator (see e.g., [19, 1]). We refer to BSTs maintained in this way as *path-balanced* BSTs (see Figure 1).

Path-balanced BSTs turn out to be surprisingly difficult to analyse. In 1995, Balasubramanian and Raman [1] showed the upper bound of $O(\log n \cdot \log\log n / \log\log\log n)$ on the cost of operations in path-balanced BSTs. This bound has not been improved since. Thus, path-balanced BSTs are not known to match the $O(\log n)$ amortized cost (let alone the stronger adaptive properties) of Splay. This is surprising, because broad classes of BSTs are known to match several guarantees of Splay trees [19, 2], path-balanced BSTs, however, fall outside these classes.[6] Without evidence to the contrary, one may even conjecture path-balanced BSTs to achieve dynamic optimality; yet our current upper bounds do not even match those of a *static* balanced tree. This points to a large gap in our understanding of a natural heuristic in the fundamental BST model.

[5] The restriction to touch only the search path is natural, as the cost of doing this is proportional to the *search cost*. (A BST can be changed into any other BST with a linear number of rotations [16].)

[6] Intuitively, path-balance is different, and more difficult to analyse than Splay, because it may increase the depth of a node by an additive $O(\log n)$, whereas Splay may increase the depth of a node by at most 2. In a precise sense, path-balance is not a *local* transformation (see [2]).

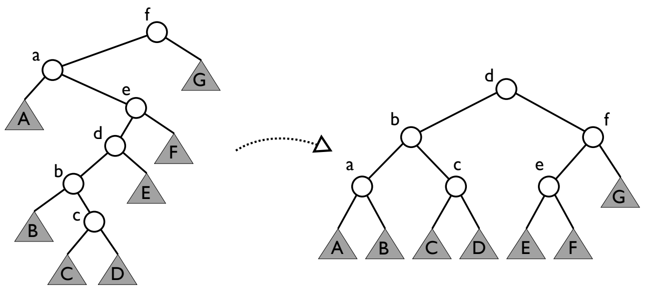


Figure 1 Access in a path-balanced BST. Search path (f, a, e, d, b, c) from root f to accessed item c is re-arranged into a balanced tree with subtrees (denoted by capital letters) re-attached.

In this paper we show that the amortized time of an access[7] in a path-balanced BST is $O\left(\log n \cdot (\log^* n)^2 \cdot 2^{\log^* n}\right)$. The result, probably not tight, comes close to the information-theoretic lower bound of $\Omega(\log n)$. Closing the gap remains a challenging open problem.

Priority queues support the operations *insert*, *delete-min*, and possibly *meld*, *decrease-key* and others. Pairing heaps, a popular priority queue implementation, were proposed in the 1980s by Fredman, Sedgewick, Sleator, and Tarjan [5] as a simpler, self-adjusting alternative to Fibonacci heaps [6]. Pairing heaps maintain a multi-ary tree whose nodes (each with an associated key) are in heap order. Similarly to Splay trees, pairing heaps only perform key-comparisons and simple local transformations on the underlying tree, with no auxiliary data stored. Fredman et al. showed that in the standard pairing heap all priority queue operations take $O(\log n)$ time. They also proposed a number of variants, including the particularly natural *multipass pairing heap*. In multipass pairing heaps, the crucial *delete-min* operation is implemented as follows. After the root of the heap (i.e., the minimum) is deleted, repeated pairing rounds are performed on the new top-level roots, reducing their number until a single root remains. In each pairing round, neighboring pairs of nodes are *linked*. Linking two nodes makes the one with the larger key the *leftmost* child of the other (Figure 2).

Pairing heaps perform well in practice [18, 14, 12]. However, Fredman [4] showed that all of their standard variants (including the multipass described above) fall short of matching the theoretical guarantees of Fibonacci heaps (in particular, assuming $O(\log n)$ cost for delete-min, the average cost of *decrease-key* may be $\Omega(\log \log n)$, in contrast to the $O(1)$ guarantee for Fibonacci heaps). The exact complexity of the standard pairing heap on sequences of intermixed delete-min, insert, and decrease-key operations remains an intriguing open problem, with significant progress through the years (see e.g., [8, 15]). However, for the multipass variant, even the basic question of whether deleting the minimum takes $O(\log n)$ amortized time remains open, the best upper bound to date being the $O(\log n \cdot \log \log n / \log \log \log n)$ originally shown by Fredman et al. Similarly to the case of path-balanced BSTs, we have thus a basic combinatorial transformation on trees, whose complexity is not well-understood.

In this paper we show that in multipass pairing heaps delete-min[8] takes amortized time $O\left(\log n \cdot \log^* n \cdot 2^{\log^* n}\right)$, the first improvement since the original paper of Fredman et al. The improvement is, from a practical perspective, not significant. Nonetheless, it reduces the gap to the theoretical optimum from $(\approx \log^{(2)} n)$ to less than $\log^{(k)} n$ for any fixed k.

[7] We only focus on successful search operations (i.e., accesses). The results can be extended to other operations at the cost of technicalities. For simplicity, we assume that the keys in the tree are unique.
[8] To keep the presentation simpler, we only focus on *delete-min* operations, omitting the extension of the result to other operations.

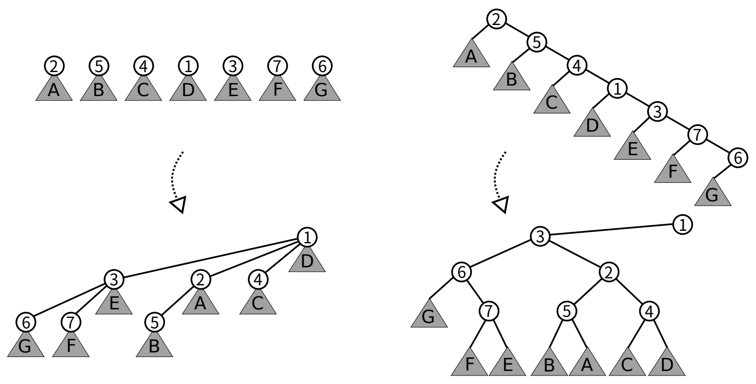


Figure 2 Delete-min in a multipass pairing heap. (above) state after deleting the root, with list of siblings; (below) state after three pairing rounds, with links $(2, 5)$, $(4, 1)$, $(3, 7)$, $(2, 1)$, $(3, 6)$, $(1, 3)$. (left) multi-ary view; (right) binary view. Numbers denote keys, capital letters denote subtrees.

The reader may notice that the old bounds for multipass pairing heaps and path-balanced BSTs are the same. The two data structures are, indeed, quite similar: if one views multipass pairing heaps as binary trees (see e.g., [10, § 2.3.2]), the multipass re-adjustement is equivalent to balancing the right-spine of a binary tree.[9] The multipass analysis, however, does not immediately transfer to path-balanced BSTs; the fact that the BST search path may be arbitrary (not necessarily right-leaning) complicates the argument for path-balanced BSTs.

Our analysis of multipass pairing heaps (§ 2) is based on a new, fine-grained scaling of the sum-of-logs potential function used by Sleator and Tarjan in the analysis of Splay trees, and by Fredman et al. in the analysis of pairing heaps. At a high level, we argue that certain link operations are information-theoretically efficient, and that such links happen sufficiently often. The subsequent, rather intricate analysis notwithstanding, we believe that the ideas of the proof may have further applications in the analysis of data structures.

In § 3 we show our result for path-balanced BSTs. Informally, we decompose the path-balancing operation into several stages, each of which resembles the multipass transformation, allowing us to adapt and reuse the result of § 2. For lack of space, we omit several proofs (marked $\star$) in this version of the paper and refer to the longer preprint[10] for details and additional illustrations.

2 Multipass pairing heaps

A *pairing heap* is a multi-ary heap, storing a key in each node, with the regular (min)heap-condition: the key of a node is smaller than the keys of its children. Priority queue operations are implemented using the unit-cost *linking* step. Given nodes x, y, $\mathsf{link}(x, y)$ "hangs" the node with the larger key as the *leftmost* child of the other. The operations *insert*, *meld*, and *decrease-key* can be implemented in a straightforward way using a single *link* (we refer to [5] for details). The only nontrivial operation is *delete-min*. Here, after deleting the root, we are left with a number of top-level nodes, which we combine into a single tree via a sequence of *links*. In multipass pairing heaps we achieve this by performing repeated *pairing rounds*, until a single top-level node remains (i.e., the new root of the heap). A single pairing round

[9] We note that the previous analysis of path-balanced BSTs [1] did not use this correspondence. By connecting the two data structures, we also simplify (to some extent) the proof of [1].
[10] `https://arxiv.org/abs/1806.08692`

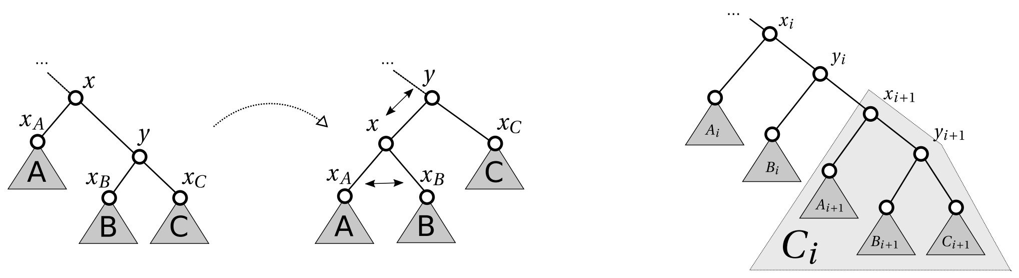


Figure 3 Left: $\mathsf{link}(x, y)$ in binary tree view. Dots (...) indicate the sequence of nodes that have already been linked in the current round, subtree C contains the yet-to-be-linked nodes. Arrows indicate possible switching depending on the outcome of the comparison between x and y. The roots of A,B, and C are denoted x_A, x_B, and x_C. Right: i-th link in a round (between x_i and y_i). The subtree rooted at the right child of y_i is denoted C_i; observe that C_i contains C_{i+1}.

is as follows. Let $x_1, \dots, x_\ell$ be the top-level nodes, ordered left-to-right, before the round. For all $1 \le i \le \lfloor \ell/2 \rfloor$ we perform $\mathsf{link}(x_{2i-1}, x_{2i})$. Observe that if ℓ is odd, then the rightmost node is unaffected in the current round. The number of rounds is $\lceil \log(k) \rceil$, where k is the number of children of the (deleted) root.[11] (See Figure 2.)

We now analyse delete-min operations implemented by multipass pairing heaps. Let k be the number of children of the deleted root, defined to be the real cost of the operation (observe that the number of links is exactly $k - 1$). Let n be the size of the heap before the operation. We use the binary tree view of multi-ary heaps, where the *leftmost child* and *next sibling* pointers are interpreted as *left child* and *right child.* A single link operation is shown in Figure 3. Let a, b, c denote the sizes of subtrees A, B, and C, respectively.

We define a potential function that refines the Sleator-Tarjan “sum-of-logs” potential [17]. Let $\Phi = \sum_{x \in T} \phi(x)$, over all nodes x of the heap T, where

$$\phi(x) = \frac{H(x)}{\log^2(2 + H(x))}, \quad \text{and} \quad H(x) = \log\left(\frac{s(p(x))}{s(x)}\right),$$

where $s(x)$ denotes the size of the subtree rooted at x, and $p(x)$ is the parent of x.[12] Note that both *subtrees* and *parents* are meant in the binary tree view.

For convenience, define the functions

$$f(x) = \log x / \log^2(2 + \log x), \quad \text{and} \quad g(x) = x / \log^2(2 + x).$$

With this notation, $f(x) = g(\log(x))$, and $\phi(x) = f\left(\frac{s(p(x))}{s(x)}\right)$. Clearly, both $f(x)$ and $g(x)$ are positive, monotone increasing, and concave, for all $x \ge 1$, respectively, $x \ge 0$.

By simple arithmetic, the increase in potential due to a single link (as in Figure 3) is:

$$\begin{aligned}\Delta\Phi \;=\; & f\left(\frac{a+b+1}{a}\right) + f\left(\frac{a+b+1}{b}\right) + f\left(\frac{a+b+c+2}{a+b+1}\right) + f\left(\frac{a+b+c+2}{c}\right) \\ & - f\left(\frac{a+b+c+2}{a}\right) - f\left(\frac{a+b+c+2}{b+c+1}\right) - f\left(\frac{b+c+1}{b}\right) - f\left(\frac{b+c+1}{c}\right). \quad (1)\end{aligned}$$

[11] The function $\log(\cdot)$ is base 2 everywhere, the base e logarithm is written as $\ln(\cdot)$.
[12] Using $\phi(x) = H(X)$ instead, would essentially recover the original “sum-of-logs” potential. Such an “edge-based” potential function was used earlier, e.g., in [7, 13].

For a suitably large constant γ (for concreteness let $\gamma = 3000$), we consider the quantities $\gamma^2 a$, γb, and c, i.e., the scaled sizes of the subtrees A, B, and C. We distinguish different kinds of links, depending on the ordering of the three quantities (breaking ties arbitrarily). We first look at the cases when $\gamma^2 a$ or γb is the largest (called respectively type-(1) and type-(2) links), and show that the possible increase in potential due to such links is small. In particular, for type-(1) links, $\Delta\Phi$ is dominated by a term $f(a/c)$, and for type-(2) links the positive and negative contributions cancel out, leaving $\Delta\Phi = O(1)$. The proofs (omitted here) use standard (although somewhat delicate) analysis.

▶ **Lemma 1** ($\star$)**.** *A type-(1) link ($\gamma^2 a \geq \max\{\gamma b, c\}$) increases the potential Φ by at most $2 \cdot g\big(\log(a/c) + O(1)\big)$, where the $O(1)$ term is a constant independent of a, b, c, n, and k.*

▶ **Lemma 2** ($\star$)**.** *A type-(2) link ($\gamma b \geq \max\{\gamma^2 a, c\}$) increases Φ by at most $O(1)$.*

The case when c is the greatest of the three quantities (called type-(3) link) is the most favorable. Here, the potential of x_A, x_B before the linking is (roughly) the logarithm of $s(x_C)$ (very large) divided by $s(x_A), s(x_B)$; after the linking, the potential becomes (essentially) the logarithm of the ratio between $s(x_A)$ and $s(x_B)$ (much smaller), resulting in a significant saving in potential. We use this saving to “pay” for the operations. First we make the following, easier claim.

▶ **Lemma 3** ($\star$)**.** *A type-(3) link ($c \geq \max\{\gamma^2 a, \gamma b\}$) can not increase Φ.*

It remains to balance the *decrease* in potential due to type-(3) links and the *increase* in potential due to all other links. First, we show that almost all links are type-(3).

▶ **Lemma 4.** *There are at most $O(\log n)$ type-(1) and type-(2) links within a pairing round.*

Proof. Let a_i, b_i, c_i denote the subtree-sizes corresponding to the i-th link *from left to right*, see Figure 3(right). Let the subsequences a_{i_t}, b_{i_t}, c_{i_t}, $t = 1, \dots, m$ be the subtree-sizes corresponding to type-(1) and type-(2) links. Observe that $c_{i_1} \geq \cdots \geq c_{i_m}$. If the i-th link is of type-(1) or type-(2), then $c_{i-1} = 2 + a_i + b_i + c_i \geq (1 + 1/\gamma^2) \cdot c_i$, since in each of these cases $a_i \geq 1/\gamma^2 c_i$ or $b_i \geq 1/\gamma^2 c_i$. Since $c_{i_1} \leq n$, and $c_{i_m} \geq 1$ the claim follows. ◀

▶ **Lemma 5.** *All type-(1) and type-(2) links within a single pairing round increase the potential by at most $O(\log n)$.*

Proof. Look at a single round of pairing. Let a_{i_t}, b_{i_t}, c_{i_t} ($t = 1, \dots, m$) be as in the proof of Lemma 4 and recall that $m = O(\log n)$. If the i_t-th link is type-(1), then by Lemma 1, the increase in potential is at most $2 \cdot g\big(\log(a_{i_t}/c_{i_t}) + O(1)\big)$.

Otherwise, if the i_t-th link is type-(2), then by Lemma 2, the increase in potential is at most $O(1)$, which we can write as $2 \cdot g(c')$, for a suitable constant c'.

Let q_t denote $\log(a_{i_t}/c_{i_t}) + O(1)$, or c', corresponding to the i_t-th link (according to its type). We have $\sum q_i \leq \alpha \cdot \log n$ (for a fixed constant $\alpha \geq 1$), since the sum of the $\log(a_i/c_i)$ terms telescopes, and the additive $O(1)$ (or c') terms appear at most $m = O(\log n)$ times.

The total increase in potential is at most $\Delta\Phi = 2 \cdot \sum_{t=1}^{m} g(q_t)$. By the concavity of $g(\cdot)$, $\Delta\Phi$ is maximized if all of the arguments of $g(\cdot)$ are equal. We thus obtain a bound on the total increase in potential in the pairing round.

$$\Delta\Phi \leq 2m \cdot g\left(\frac{\alpha \cdot \log n}{m}\right) = \frac{2\alpha \log n}{\log^2\left(2 + \alpha \cdot (\log n)/m\right)} = O\left(\log n\right). \quad ◀$$

The last proof yields, in fact, the following stronger claim.

▶ **Lemma 6.** *All type-(1) and type-(2) links within the last* $(\log\log n)$ *pairing rounds increase the potential by at most* $O(\log n)$.

Proof. Observe that for $j < \log\log n$, the j-th to the last pairing round has at most $m \le 2^j < \log n$ links. Thus, as in Lemma 5, we obtain:

$$\Delta\Phi \le \frac{2\alpha\log n}{\log^2\left(2+\alpha\cdot(\log n)/m\right)} \le \frac{2\alpha\log n}{\log^2\left(\alpha\cdot(\log n)/2^j\right)} = \frac{2\alpha\log n}{((\log\log n+\log\alpha)-j)^2}.$$

Note that the second inequality holds since $2^j < \log n$. The sum of this expression over all $(\log\log n)$ levels j is $O(\log n)$. (Using the fact that $\sum_k 1/k^2$ converges to a constant.) ◀

Now we estimate more carefully the decrease in potential due to type-(3) links. Let x_A and x_B be nodes as denoted in Figure 3. We want to express the potential-change in terms of $H_A = H(x_A)$ and $H_B = H(x_B)$ (before the link operation). Recall that $H_A = \log\left(\frac{a+b+c+2}{a}\right)$ and $H_B = \log\left(\frac{b+c+1}{b}\right)$.

Among type-(3) links ($c \ge \max\{\gamma^2 a, \gamma b\}$) we distinguish two subtypes: type-(3A) ($\gamma^2 a \ge \gamma b$), and type-(3B) ($\gamma b \ge \gamma^2 a$). We have the following two (symmetric) observations:

▶ **Lemma 7** ($\star$). *A type-(3A) link* ($c \ge \gamma^2 a \ge \gamma b$) *decreases the potential by at least*

$$\Omega(1)\cdot\frac{H_A}{\log^2(2+H_B)} - O(1).$$

It follows that for some constant d_1, if $H_A \ge d_1\cdot\log^2(2+H_B)$, then $\Delta\Phi \le -1$.

▶ **Lemma 8** ($\star$). *A type-(3B) link* ($c \ge \gamma b \ge \gamma^2 a$) *decreases the potential by at least*

$$\Omega(1)\cdot\frac{H_B}{\log^2(2+H_A)} - O(1).$$

It follows that for some constant d_2, if $H_B \ge d_2\cdot\log^2(2+H_A)$, then $\Delta\Phi \le -1$.

▶ **Corollary 9.** *There exists a constant* d $(=\max(d_1,d_2))$ *such that all type-(3A) links with* $H_A \ge d\cdot\log^2(2+H_B)$ *and all type-(3B) links with* $H_B \ge d\cdot\log^2(2+H_A)$ *decrease the potential by at least* 1.

We now define the *category* of a node with respect to its $H(\cdot)$ value. Intuitively, nodes of the same category are those that, when linked, release the most potential. Let us denote $h(x) = d\cdot\log^2(2+x)$. Using the notation of function composition, let

$$h^{(0)}(x) = x,\quad h^{(i)}(x) = h\left(h^{(i-1)}(x)\right).$$

The category of a node is based on the values $h^{(i)}(\log n)$, $i = 1,\dots,\log^* n$. Note that $h^{(0)}(\log n) = \log n$, $h^{(1)}(\log n) = d\cdot\log^2(2+\log n),\dots,h^{(\log^* n)}(\log n) = O(1)$, where the $O(1)$ depends on d, since (using the *star* notation) $h^*(n) \le \left(\log^3\right)^*(n)+O(1) = \log^* n+O(1)$.

▶ **Definition 10** (Category). Let u be a node. For $i = 1,\dots,\log^* n$, we let $\mathsf{cat}(u) = i$ if:

$$H(u.\mathsf{left}) \in (h^{(i)}(\log n), h^{(i-1)}(\log n)].$$

If $H(u.\mathsf{left}) \le h^{(\log^* n)}(\log n)$ we say that u is of category 0.

The following crucial observations connect categories and savings in potential.

▶ **Lemma 11.** *Let* $\mathit{link}(u,v)$ *be type-(3). If* $\mathsf{cat}(u)=\mathsf{cat}(v)\neq 0$*, then the link decreases the potential by at least* 1*.*

Proof. Note that if $i=\mathsf{cat}(u)=\mathsf{cat}(v)\neq 0$ then

$$H(u.\mathsf{left})\geq h^{(i)}(\log n)\geq d\cdot\log^2(2+H(v.\mathsf{left})),$$

$$H(v.\mathsf{left})\geq h^{(i)}(\log n)\geq d\cdot\log^2(2+H(u.\mathsf{left})).$$

Thus, by Corollary 9, the claim follows. ◀

▶ **Lemma 12.** *In each pairing round there are at most* $O(\log n)$ *nodes of category* 0*.*

Proof. Let x be of category 0, then $H(x.\mathsf{left})=O(1)$. Denoting $a=s(x.\mathsf{left})$, $c=s(x.\mathsf{right})$, we get $H(x.\mathsf{left})=\log\frac{a+c+1}{a}=O(1)$. Therefore, $a=\Omega(c)$, an occurrence that can happen at most $O(\log n)$ times in each round (by the same argument as in Lemma 4). ◀

▶ **Lemma 13.** *Let* w *denote the "winner" of linking* x *and* y *(neither of category* 0*), i.e.,* w *is the one with the smaller key. Then* $\mathsf{cat}(w)\geq\max\{\mathsf{cat}(x),\mathsf{cat}(y)\}$*.*

Proof. Let $y=x.\mathsf{right}$, $a=s(x.\mathsf{left}), b=s(y.\mathsf{left}), c=s(y.\mathsf{right})$ as in Figure 3. We have that $H(x.\mathsf{left})=\log\frac{a+b+c+2}{a}$, $H(y.\mathsf{left})=\log\frac{b+c+1}{b}$, and $H(\mathsf{link}(x,y).\mathsf{left})=\log\frac{a+b+c+2}{a+b+1}$.

Clearly $\frac{a+b+c+2}{a+b+1}\leq\min\{\frac{a+b+c+2}{a},\frac{b+c+1}{b}\}$, finishing the proof. ◀

As seen in Figure 2, a delete-min operation transforms the "spine" of the heap (in binary view) into a balanced tree. We denote this tree by T. Each level of T corresponds to a pairing round; specifically, level i of T consists of nodes at distance i from the leaves, containing the *losers* of the i-th pairing round. The following lemma captures the potential reduction that yields the main result.

▶ **Lemma 14.** *Let* T' *be a subtree of* T *of depth* $\log^* n$*, whose leaves correspond to* $2^{\log^* n}$ *consecutive link operations. If* T' *contains only type-(3) links and no links involving nodes of category* 0*, then the total decrease in potential caused by the links of* T' *is at least* 1*.*

Proof. Assume towards contradiction that there is no link between two nodes of the same category in T'. By Lemma 13 in each round the minimal overall category increases by at least 1, leaving us with two nodes of maximal category in the last round, a contradiction. By Lemma 11, a link between nodes of equal category decreases the potential by at least 1. ◀

▶ **Theorem 15.** *The amortized time of delete-min in multipass pairing heaps is* $O(\log n\cdot\log^* n\cdot 2^{\log^* n})$*.*

Proof. Let the real cost (number of link operations) be k. Note that there are at most $\lceil\log k\rceil$ pairing rounds.

Thus, if $k\leq\log n\cdot\log^* n\cdot 2^{\log^* n}$, then there are at most $\log\log n+\log\log^* n+\log^* n+1$ rounds. Using Lemma 5 we get that the first $\log\log^* n+\log^* n+1=O(\log^* n)$ pairing rounds increase the potential by at most $O(\log n\cdot\log^* n)$. Also, as shown in Lemma 6, the total increase in potential for the last $\log\log n$ levels is $O(\log n)$. Thus, the total potential increase is at most $O(\log n)+O(\log n\cdot\log^* n)$.

To analyse the case $k>\log n\cdot\log^* n\cdot 2^{\log^* n}$, we use the potential decrease of type-(3) links. First, we look at the first $\log^* n$ pairing rounds.

By Lemma 14, the links in every complete subtree of T of depth $\log^* n$, in which there are only type-(3) links and no category-0 nodes, decrease the potential by at least 1.

In the first $\log^* n$ levels of T we can find $\frac{k}{2^{\log^* n}}$ *disjoint* subtrees of this size. In these levels there are at most $O(\log^* n \cdot \log n)$ type-(1),(2) links, or links containing category-0 nodes (Lemmas 4 and 12). Thus, at least $\frac{k}{2^{\log^* n}} - O(\log^* n \cdot \log n)$ of the subtrees answer the conditions of Lemma 14, decreasing the potential by at least $\frac{k}{2^{\log^* n}} - O(\log^* n \cdot \log n)$. Also, the total increase in potential caused by type-(1),(2) links is at most $O(\log n \cdot \log^* n)$ (Lemma 5). Therefore, the first $\log^* n$ levels give us a decrease in potential of at least $\frac{k}{2^{\log^* n}} - O(\log^* n \cdot \log n)$.

Note that by using the same argument on the next $\log^* n$ levels, we get a decrease in potential of at least $\frac{k'}{2^{\log^* n}} - O(\log^* n \cdot \log n)$, where k' is the number of links in level $\log^* n+1$. Thus, levels which contain $\Omega\left(\log n \cdot \log^* n \cdot 2^{\log^* n}\right)$ links only decrease the potential.

We repeat this argument until we reach a level in T containing $\tilde{k} \le \log n \cdot \log^* n \cdot 2^{\log^* n}$ links. Now, applying the same argument as for the first case, we get that the total increase in potential for the last $\log \tilde{k}$ levels (starting from the level of $\tilde{k}$ links) is at most $O(\log n \cdot \log^* n)$.

Summarizing, the total amortized time (in both cases) is at most

$$k + O(\log n \cdot \log^* n) - \left(\frac{k}{2^{\log^* n}} - \log^* n \cdot \log n\right).$$

Scaling the potential by $2^{\log^* n}$, we get that the amortized time is $O(\log n \cdot \log^* n \cdot 2^{\log^* n})$. ◀

3 Path-balanced binary search trees

Consider the operation of accessing a node x in a BST T with n nodes (we refer interchangeably to a node and its key). Let $\mathcal{P}^x$ denote the search path to x (i.e., the path from the root of T to x). The path-balance method re-arranges $\mathcal{P}^x$ into a complete balanced BST (with all levels complete, except possibly the lowest). Subtrees hanging off $\mathcal{P}^x$ are re-attached in the unique way given by the key-order (Figure 1). There are multiple ways to implement this transformation such that the number of pointer moves and pointer changes is linear in the length of the search path. For instance, we may first rotate the search path into a *monotone* path, then apply a *multipass transformation* (described next) to this monotone path.

Multipass transformation. A multipass transformation of a monotone path $\mathcal{P}$ (of which the deepest node might not be a leaf) converts $\mathcal{P}$ into a balanced tree (in which the last level may be incomplete) by a sequence of *pairing rounds*. In each pairing round we rotate every other edge in a prefix of $\mathcal{P}$ (i.e., a subpath of the shallowest nodes on $\mathcal{P}$). Each rotation pushes one node off $\mathcal{P}$. We denote by $\mathcal{P}^i$ the path remaining of $\mathcal{P}$ after i pairing rounds. The pairing rounds are defined as follows. We assume that the path consists of right child pointers; in the case it consists of left child pointers everything is symmetric.

Let $\ell(\mathcal{P})$ denote the length of $\mathcal{P}$ (i.e., the number of nodes on $\mathcal{P}$). In the first round we do just enough rotations so that the length of the path after the round (i.e., $\mathcal{P}^1$) is one less than a power of 2. Specifically, we do α rotations where α is the smallest integer such that $\ell(\mathcal{P}^1) = \ell(\mathcal{P}) - \alpha = 2^j - 1$. In the second round we do $2^{j-1} - 1$ rotations on $\mathcal{P}^1$, and in round $i > 1$ we do $2^{j-i+1} - 1$ rotations on $\mathcal{P}^{i-1}$. We maintain the invariant that after $i + 1$ rounds all the nodes that were pushed off $\mathcal{P}$ (excluding those that were pushed off $\mathcal{P}$ at the first round) are arranged in balanced binary trees of height $(i - 1)$, hanging as children of the nodes of $\mathcal{P}^{i+1}$.

The proof of the following theorem is analogous to the proof of Theorem 15 (one can verify that all steps of the proof still hold for the slightly modified pairing rounds of the multipass transformation, replacing rotations by links).

▶ **Theorem 16.** *For every monotone path $\mathcal{P}$ with $\ell(\mathcal{P}) = k$, the change in Φ caused by applying a multipass transformation on $\mathcal{P}$ is bounded by $\Delta\Phi \le c(n,k) := -\frac{k}{2^{\log^* n}} + O(\log n \cdot \log^* n)$, where n is the size of the subtree of the root of $\mathcal{P}$.*

Warm-up: a simplified path-balance. We first look at an easier-to-analyse variant of path-balance, where, instead of a complete balanced tree, we build an *almost* balanced tree out of the search path $\mathcal{P}^x$, as follows: we first make the accessed item x the root, then turn the parts of $\mathcal{P}^x$ containing items smaller (resp. larger) than x into balanced subtrees rooted at the left (resp. right) child of x. The depth of this tree is at most one larger than the depth of a complete balanced tree built from $\mathcal{P}^x$.

For the purpose of the analysis, we view the simplified path-balance transformation as a two-step process. The actual implementation may be different but the analysis applies as long as the transformation takes time $O(\ell(\mathcal{P}^x))$.

Step 1. Rotate the accessed element x all the way to the root. (Observe that after this step, $\mathcal{P}^x$ is split into two monotone paths, $\mathcal{P}^{<x}$ to the left of x consisting only of "right child" pointers, and $\mathcal{P}^{>x}$ to the right of x, consisting only of "left child" pointers.)

Step 2. Apply a multipass transformation to $\mathcal{P}^{>x}$ and to $\mathcal{P}^{<x}$.

We show that the amortized time of an access using simplified path-balance is $O(\log n \cdot \log^* n \cdot 2^{\log^* n})$. We use the same potential function as in § 2, and we assume the two-step implementation described above. We first state an easy observation.

▶ **Lemma 17.** *Let $\mathcal{P}$ be a path in T rooted at a node r, then $\Phi(\mathcal{P}) = O(\log s(r))$, where $\Phi(\mathcal{P}) = \sum_{x \in \mathcal{P}} \phi(x)$ and $s(r)$ is the size of the subtree of r.*

Proof. Denote $\ell = \ell(\mathcal{P})$. Let $a_1 \le ... \le a_\ell = s(r)$ be the subtree-sizes of the nodes on $\mathcal{P}$ from the deepest node to r. Then

$$\Phi(\mathcal{P}) = \sum_{k=1}^{\ell-1} f\left(\frac{a_{k+1}}{a_k}\right) = \sum_{k=1}^{\ell-1} g\left(\log \frac{a_{k+1}}{a_k}\right) \le \ell \cdot g\left(\frac{\log s(r)}{\ell}\right) = O(\log s(r)),$$

due to g's concavity and since the terms $\log \frac{a_{k+1}}{a_k}$ sum to $\log s(r) - \log a_1 \le \log s(r)$. ◀

We proceed with the analysis. We argue that rotating x to the root (Step 1) increases Φ by at most $O(\log n)$. To see this, observe first, that the potential of nodes hanged on the nodes of $\mathcal{P}^x$ excluding x, can only decrease. This is because their subtree remains the same, whereas the subtree of their parent (a node on the search path) can only lose elements. The two children of x may increase the potential by at most $O(\log n)$.

For nodes *on the search path*, we look at the potential after the transformation. We have two separate paths, and by Lemma 17 the potential of each path is bounded by $O(\log n)$. This concludes the analysis for Step 1.

In Step 2, as we apply the multipass transformation to both $\mathcal{P}^{<x}$ and $\mathcal{P}^{>x}$, Theorem 16 applies. Thus, $\Delta\Phi$ is at most $c(s(x.\mathsf{left}), \ell(\mathcal{P}^{<x})) + c(s(x.\mathsf{right}), \ell(\mathcal{P}^{>x}))$ where $c(n,k)$ is defined in Theorem 16. The claim on the amortized running time follows by scaling $\Delta\Phi$ by $2^{\log^* n}$ and adding it to the actual cost (the length of $\mathcal{P}^x$). This concludes the proof.

Analysis of path-balance. The original path-balance heuristic (where we insist on building a *complete* balanced tree) is trickier to analyse. Here, instead of moving the accessed item x to the root, we move the *median* item m of the search path $\mathcal{P}^x$ to the root. Here, "median" is meant with respect to the ordering of keys; m is, in general, *not* the node with median depth on $\mathcal{P}^x$. It is instructive to prove the earlier $O(\log n \cdot \log\log n / \log\log\log n)$ result first, by re-using parts of the Fredman et al. proof for multipass. We defer this to the full version of the paper. In the remainder of this section we prove the new, stronger result.

▶ **Theorem 18.** *The amortized time of search in a path-balanced BST of size n is* $O\left(\log n \cdot (\log^* n)^2 \cdot 2^{\log^* n}\right)$.

For the purpose of the analysis, we view the path-balance transformation as a sequence of recursive calls on search paths in *some* subtree of T. The total *real* cost is proportional to the original length of the search path to x which we denote by k. We define a threshold $\tau = \log n$, and distinguish between recursive calls on paths shorter than τ ("short paths") and recursive calls on paths longer than τ ("long paths").

A **long path** $\mathcal{P}^x$ is processed as follows. Rotate the median m of the nodes on $\mathcal{P}^x$ to the root, splitting $\mathcal{P}^x$ into two paths of equal lengths. One of these paths contains the path from m to x in $\mathcal{P}^x$, and the other path, which is monotone, contains either the elements smaller than m on $\mathcal{P}^x$ or the elements larger than m on $\mathcal{P}^x$ (depending on whether x is in the right or left subtree of m). In the sequel we assume without loss of generality that the monotone part contains all elements larger than m and denote it by $\mathcal{P}^{>m}$. Let Q^x denote the other (non-monotone) path that ends with x. We perform a multipass transformation on $\mathcal{P}^{>m}$, and make a recursive call on Q^x (i.e., Q^x becomes the P^x of the next recursive call).

A **short path** $\mathcal{P}^x$ is transformed into a balanced binary tree in two phases, as follows. In the first phase, rotate up the median m_1 of $\mathcal{P}^x = \mathcal{P}^1$ until it becomes the root of the subtree rooted at the shallowest node of $\mathcal{P}^1$. This decomposes $\mathcal{P}^1$ into a monotone path and a general path $\mathcal{P}^2$, one starting at the left child of m_1 and the other at the right child of m_1. We repeat this recursively with the median m_2 of $\mathcal{P}^2$, and so on, until we get a general path $\mathcal{P}^\ell$ of length 1. After this transformation, the medians m_j form a path, each m_j having the next median m_{j+1} as one child and a monotone path as the other child. The lengths of these monotone paths decrease exponentially by a factor of 2. In the second phase we apply a multipass transformation on each monotone path, obtaining a complete balanced tree.

Before we analyse each case, we argue that Theorem 16 also holds with a modified potential Φ (defined below). As we only use the new potential from now on, there is no risk of confusion. The modification consists in changing the exponent of the logarithmic term in the denominator from 2 to 3, and changing the additive constant inside the $\log(\cdot)$ to make sure Φ is still increasing everywhere.

Formally, $\Phi = \sum_{x \in T} \phi(x)$, where $\phi(x) = \frac{H(x)}{\log^3(4+H(x))}$, and $H(x) = \log \frac{s(p(x))}{s(x)}$. As earlier, $s(x)$ is the size of the subtree rooted at x, and $p(x)$ is the parent of x.

It can be shown that the entire analysis in § 2 extends to this new potential. Therefore, Theorem 16 holds also for the modified potential function Φ. Now, the analysis of transforming long paths is straightforward. For short paths, we need two new observations.

▶ **Lemma 19.** *The total increase in potential for performing multipass transformation on a path $\mathcal{P}$ of length $k < \log n$ where n is the size of the subtree of the root of $\mathcal{P}$, is at most*

$$\sum_{j=1}^{\log k} \frac{O(\log n)}{(\log\log n + 1 - j)^3}.$$

The proof is identical to that of Lemma 6. As before, the sum can be bounded as $O(\log n)$, but here we use the quantity explicitly inside another sum where the exponent 3 in the denominator will be crucial. The next observation can be shown in a way similar to Lemma 17.

▶ **Lemma 20** ($\star$). *Given a search path $\mathcal{P}$ of length $k < \log n$, the total increase in Φ due to recursively rotating all medians $m_1, m_2, \ldots$ of $\mathcal{P}$ to the root is $O(\log n)$.*

We are ready to prove Theorem 18. We split the proof into three cases according to the length of the search path, denoted by k.

Short paths ($k \le \tau = \log n$). Notice that $\log k \le \log\log n$. Recall that in the first phase, we repeatedly rotate up the medians, decomposing the path into monotone paths of lengths $1, 2, 4, \ldots, 2^j$, where $j < \log\log n$. By Lemma 20 the total increase in potential due to this transformation is at most $O(\log n)$.

In the second phase, we do a multipass transformation on each of these monotone paths. By Lemma 19, a multipass transformation on a monotone path of length 2^j increases Φ by at most $\sum_{i=1}^{j} \alpha \cdot \log n/(\log\log n + 1 - i)^3$, for some fixed α. Thus, the $j < \log\log n$ multipass transformations increase the potential by at most

$$\sum_{j=1}^{\log\log n} \sum_{i=1}^{j} \frac{\alpha \cdot \log n}{(\log\log n + 1 - i)^3} = \sum_{s=1}^{\log\log n} \frac{\alpha \cdot \log n}{s^2} = O(\log n).$$

The first equality holds since the term $\frac{\alpha \cdot \log n}{s^3}$ appears in the above sum exactly s times ($1 \le s \le \log\log n$). Thus, the total increase in Φ is, in this case, $O(\log n)$.

Longish paths ($\tau < k \le \log n \cdot \log^* n \cdot 2^{\log^* n}$). Notice that $\log k \le \log\log n + 2 \cdot \log^* n$.

We perform $2 \cdot \log^* n$ recursive calls and a final call on a search path of length $k' \le \tau$. The final call increases Φ by at most $O(\log n)$, by the analysis in the previous case. The recursive calls consist of rotating the current median up to the root and applying the multipass transformation on a monotone path. As before, rotating the median up increases Φ by at most $O(\log n)$. Also, each multipass transformation is performed on a path of length $\le \log n \cdot \log^* n \cdot 2^{\log^* n}$. By Theorem 16, the increase in potential is at most $O(\log n \cdot \log^* n)$. Therefore, the $2 \cdot \log^* n$ recursive calls increase Φ by at most $O\left(\log n \cdot (\log^* n)^2\right)$, which also bounds the total increase in Φ.

Long paths ($k = \Omega\left(\log n \cdot \log^* n \cdot 2^{\log^* n}\right)$). We look at the potential change due to the first recursive call. Again, rotating the median m to the root increases Φ by at most $O(\log n)$. The path splits into $\mathcal{P}^{>m}$ and Q^x, of which $\mathcal{P}^{>m}$ is monotone. By Theorem 16, the multipass transformation on $\mathcal{P}^{>m}$ *decreases* Φ by $\frac{k/2}{2^{\log^*(n)}} - O\left(\log^*(n) \cdot \log n\right)$.

By the same argument, Φ decreases during all of the subsequent recursive calls on paths of size $\Omega\left(\log n \cdot \log^* n \cdot 2^{\log^* n}\right)$.

We continue until we have a recursive call on a path of size at most $\left(\log n \cdot \log^* n \cdot 2^{\log^* n}\right)$, which, by the previous case, increases Φ by at most $O\left(\log n \cdot (\log^* n)^2\right)$. Thus, we obtain that the total decrease in Φ in this case is at least $\frac{k/2}{2^{\log^*(n)}} - O\left(\log n \cdot (\log^* n)^2\right)$.

Combining the three cases, after scaling the potential by $2 \cdot 2^{\log^* n}$, we conclude that the amortized time of the access is $k + 2 \cdot 2^{\log^* n} \cdot \Delta\Phi = O\left(\log n \cdot 2^{\log^* n} \cdot (\log^* n)^2\right)$, as required.

References

1 R. Balasubramanian and Venkatesh Raman. Path balance heuristic for self-adjusting binary search trees. In *Proceedings of FSTTCS*, pages 338–348, 1995. doi:10.1007/3-540-60692-0_59.

2 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Self-adjusting binary search trees: What makes them tick? In *ESA 2015*, pages 300–312, 2015. doi:10.1007/978-3-662-48350-3_26.

3 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. The landscape of bounds for binary search trees. *CoRR*, abs/1603.04892, 2016. arXiv:1603.04892.

4 Michael L. Fredman. On the efficiency of pairing heaps and related data structures. *J. ACM*, 46(4):473–501, 1999. doi:10.1145/320211.320214.

5 Michael L. Fredman, Robert Sedgewick, Daniel Dominic Sleator, and Robert Endre Tarjan. The pairing heap: A new form of self-adjusting heap. *Algorithmica*, 1(1):111–129, 1986. doi:10.1007/BF01840439.

6 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*, pages 338–346, 1984. doi:10.1109/SFCS.1984.715934.

7 George F. Georgakopoulos and David J. McClurkin. Generalized template splay: A basic theory and calculus. *Comput. J.*, 47(1):10–19, 2004. doi:10.1093/comjnl/47.1.10.

8 John Iacono. Improved upper bounds for pairing heaps. In *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings*, pages 32–45, 2000. doi:10.1007/3-540-44985-X_5.

9 John Iacono. In pursuit of the dynamic optimality conjecture. In *Space-Efficient Data Structures, Streams, and Algorithms*, volume 8066 of *Lecture Notes in Computer Science*, pages 236–250. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-40273-9_16.

10 Donald E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1997.

11 Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

12 Daniel H. Larkin, Siddhartha Sen, and Robert Endre Tarjan. A back-to-basics empirical study of priority queues. In *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2014, Portland, Oregon, USA, January 5, 2014*, pages 61–72, 2014. doi:10.1137/1.9781611973198.7.

13 Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*, volume 1 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1984. doi:10.1007/978-3-642-69672-5.

14 Bernard M. E. Moret and Henry D. Shapiro. *An empirical analysis of algorithms for constructing a minimum spanning tree*, pages 400–411. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991. doi:10.1007/BFb0028279.

15 Seth Pettie. Towards a final analysis of pairing heaps. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 174–183, 2005. doi:10.1109/SFCS.2005.75.

16 Daniel D. Sleator, William P. Thurston, and Robert Endre Tarjan. Rotation distance,triangulations,and hyperbolic geometry. Technical Report CS-TR-131-88, Princeton University (NJ US), 1988. URL: http://opac.inria.fr/record=b1019357.

17 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985. doi:10.1145/3828.3835.

18 John T. Stasko and Jeffrey Scott Vitter. Pairing heaps: Experiments and analysis. *Commun. ACM*, 30(3):234–249, 1987. doi:10.1145/214748.214759.

19 Ashok Subramanian. An explanation of splaying. *J. Algorithms*, 20(3):512–525, 1996. doi:10.1006/jagm.1996.0025.

Improved Time and Space Bounds for Dynamic Range Mode

Hicham El-Zein
Cheriton School of Computer Science, University of Waterloo, Canada
helzein@uwaterloo.ca

Meng He
Faculty of Computer Science, Dalhousie University, Canada
mhe@cs.dal.ca

J. Ian Munro
Cheriton School of Computer Science, University of Waterloo, Canada
imunro@uwaterloo.ca

Bryce Sandlund
Cheriton School of Computer Science, University of Waterloo, Canada
bcsandlund@uwaterloo.ca

Abstract

Given an array A of n elements, we wish to support queries for the most frequent and least frequent element in a subrange $[l, r]$ of A. We also wish to support updates that change a particular element at index i or insert/ delete an element at index i. For the range mode problem, our data structure supports all operations in $O(n^{2/3})$ deterministic time using only $O(n)$ space. This improves two results by Chan et al. [3]: a linear space data structure supporting update and query operations in $\tilde{O}(n^{3/4})$ time and an $O(n^{4/3})$ space data structure supporting update and query operations in $\tilde{O}(n^{2/3})$ time. For the range least frequent problem, we address two variations. In the first, we are allowed to answer with an element of A that may not appear in the query range, and in the second, the returned element must be present in the query range. For the first variation, we develop a data structure that supports queries in $\tilde{O}(n^{2/3})$ time, updates in $O(n^{2/3})$ time, and occupies $O(n)$ space. For the second variation, we develop a Monte Carlo data structure that supports queries in $O(n^{2/3})$ time, updates in $\tilde{O}(n^{2/3})$ time, and occupies $\tilde{O}(n)$ space, but requires that updates are made independently of the results of previous queries. The Monte Carlo data structure is also capable of answering k-frequency queries; that is, the problem of finding an element of given frequency in the specified query range. Previously, no dynamic data structures were known for least frequent element or k-frequency queries.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases dynamic data structures, range query, range mode, range least frequent, range k-frequency

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.25

Related Version A full version of the paper is available at `https://arxiv.org/abs/1807.03827`.

1 Introduction

The mode of a sample is a fundamental data statistic along with median and mean. Given an ordered sequence, the range mode of interval $[l, r]$ is the mode of the subsequence from index l to r. Building a data structure to efficiently compute range modes allows data analysis to

© Hicham El-Zein, Meng He, J. Ian Munro, and Bryce Sandlund;
licensed under Creative Commons License CC-BY

Table 1 Known static and dynamic range query upper bounds.

Query Type	Query Time	Update Time	Space	Citation
Sum	$O(1)$	-	$O(n)$	trivial
	$O(\lg n)$	$O(\lg n)$	$O(n)$	[15]
Min/ Max	$O(1)$	-	$O(n)$	[9]
	$O(\lg n/\lg\lg n)$	$O(\lg^{1/2+\varepsilon} n)$	$O(n)$	[5]
Median	$O(\lg n/\lg\lg n)$	-	$O(n)$	[2]
	$O((\lg n/\lg\lg n)^2)$	$O((\lg n/\lg\lg n)^2)$	$O(n)$	[13]
Majority	$O(1)$	-	$O(n)$	[7]
	$O(\lg n/\lg\lg n)$	$O(\lg n)$	$O(n)$	[7, 10]
Mode	$O(n^\varepsilon \lg\lg n)$	-	$O(n^{2-2\varepsilon})$	[14]
	$O(1)$	-	$O(n^2 \lg\lg n/\lg n)$	[14]
	$O(\sqrt{n/\lg n})$	-	$O(n)$	[3]
	$O(n^{3/4} \lg n/\lg\lg n)$	$O(n^{3/4} \lg\lg n)$	$O(n)$	[3]
	$O(n^{2/3} \lg n/\lg\lg n)$	$O(n^{2/3} \lg n/\lg\lg n)$	$O(n^{4/3})$	[3]
	$O(n^{2/3})$	$O(n^{2/3})$	$O(n)$	new
Least Frequent	$O(\sqrt{n})$	-	$O(n)$	[4]
	$O(n^{2/3} \lg n \lg\lg n)$	$O(n^{2/3})$	$O(n)$	new
k-Frequency	$O(n^{2/3})$	$O(n^{2/3} \lg n)$	$O(n)$	new

be conducted over any window of one-dimensional data. Techniques to answer such queries are relevant to the design of database systems.

The range least frequent problem can be seen as a low-frequency variant of range mode. Instead of searching for the most frequent element in an interval of the sequence, we query for an element that occurs the fewest number of times. We may either restrict our attention to elements that occur at least once in the query range or allow the answer to be an element that occurs zero times in the interval but is present elsewhere in the sequence.

A third range frequency query we consider in this paper is the problem of identifying an element with given frequency k in the specified query interval. This problem has been called the range k-frequency problem.

Both range mode query and range least frequent query have theoretical connections to matrix multiplication. In particular, the ability to answer n range mode queries on an array of size $O(n)$ in faster than $O(n^{\omega/2})$ time, where ω is the constant in the exponent of the running time of matrix multiplication, would imply a faster algorithm for boolean matrix multiplication [3, 4]. Upon closer examination, this lower bound also applies to the range k-frequency problem.

The range mode, range least frequent, and range k-frequency problems are part of a set of questions one can ask on a subrange of a sequence. Other queries in this area include:

- Sum: Return the sum of elements in the query range.
- Min/ Max: Return the minimum/ maximum in the query range.
- Median: Determine the median element in the query range.
- Majority: Return the element that occurs more than 1/2 the time, if such an element exists.

Note that the mean of a range can be reduced to the range sum problem. Table 1 gives an overview of known upper bounds on related static and dynamic range query data structures.

1.1 Our Results

We improve the results of Chan et al. [3] by giving a dynamic range mode data structure that takes $O(n)$ space and supports updates and queries in $O(n^{2/3})$ time. This improves the query/ update time of their linear space data structure by a polynomial factor and additionally improves the query/ update time of their $O(n^{4/3})$ space data structure by a $O(\log n/\log\log n)$ factor. We also include in our update procedures the ability to insert or delete elements in the middle of the array, an operation not addressed in previous dynamic range mode data structures.

Our improvements are based on the observation that knowing how many of a type of element occur in an interval can be as valuable as knowing the elements themselves. Specifically, instead of storing the frequency counts of elements per span, we store the number of elements with a particular frequency count per span. This information can be dynamically maintained, and uses $O(\log n)$ bits per frequency per span, rather than $O(\log n)$ bits per unique element per span.

Our technique is general enough to also apply to the range least frequent problem in a dynamic setting. To our knowledge, this is the first data structure to do so. In the version of the problem where we allow an answer to not occur in the specified query interval, our data structure supports queries in $O(n^{2/3}\log n\log\log n)$ time, updates in $O(n^{2/3})$ time, and occupies $O(n)$ space. In the version where the least frequent element must be present in the query interval, we develop a Monte Carlo data structure that supports queries in $O(n^{2/3})$ time, updates in $O(n^{2/3}\log n)$ time, and occupies $O(n\log^2 n)$ space. This data structure is correct with high probability for any polynomial sequence of updates and queries, with the restriction that updates are made independently of the results of previous queries. Notably, if the set of queries and updates is fixed in advance or given to the algorithm all at once, this property holds.

Furthermore, our Monte Carlo data structure is powerful enough to apply to the dynamic range k-frequency problem, also supporting queries in $O(n^{2/3})$ time, updates in $O(n^{2/3}\log n)$ time, and occupying $O(n\log^2 n)$ space. This data structure can be augmented to count the number of elements below, above, or at a given frequency, supporting both queries and updates in $O(n^{2/3})$ time and using $O(n)$ space, without the need for an independence assumption.

We organize our results as follows. In Section 2, we review previous work on static and dynamic range mode and least frequent element queries. In Section 3, we briefly give some notation that will be used for the rest of the paper. In Section 4, we give the basic setup of the $O(n)$ space data structure. Section 5 describes how to answer a range mode query in $O(n^{2/3})$ time. Section 6 explains how to support updates to our base data structures in $O(n^{2/3})$ time. In Section 7, we discuss how to answer range least frequent queries in $\tilde{O}(n^{2/3})$ time. Section 8 describes how to find an element of a given frequency in a specified range in $O(n^{2/3})$ time. Here we also mention additional frequency operations our data structure can support. Due to space limitations, we will not describe how the data structure can be made to support insertion and deletion of elements in the conference version of this paper.

2 Previous Work

2.1 Static Range Mode Query

The static range mode query problem was first studied by Krizanc et al. [14]. Their focus is primarily on subquadratic solutions with fast queries, achieving $O(n^{2-2\varepsilon})$ space and $O(n^{\varepsilon}\log n)$ query time, with $0 < \varepsilon \leq 1/2$, and $O(n^2\log\log n/\log n)$ space and $O(1)$ query time. If we set $\varepsilon = 1/2$ with the first approach, this gives a linear space static range mode data structure with query time $O(\sqrt{n}\log n)$. By substituting an $O(\log\log n)$ data structure for predecessor search, such as van Emde Boas trees, the query time can immediately be improved to $O(\sqrt{n}\log\log n)$.

Chan et al. [3] focus on linear space solutions to static range mode. They achieve a clever array-based solution with $O(n)$ space and $O(\sqrt{n})$ query time. By using bit-packing tricks and more advanced data structures, they reduce the query time to $O(\sqrt{n/\log n})$.

As with many range query data structures, the range mode problem has also been studied in an approximate setting [12, 1].

2.2 Static Range Least Frequent Query

The range least frequent problem was first studied by Chan et al. [4]. They again focus on linear-space solutions, this time achieving $O(n)$ space and $O(\sqrt{n})$ time query. In their paper, they focus on the version of range least frequent element where the element must occur in the query range.

2.3 Dynamic Range Mode

Chan et al. [3] also study the dynamic range mode problem. They give a solution tradeoff that at linear space, achieves $O(n^{3/4}\log n/\log\log n)$ worst-case time range mode query and $O(n^{3/4}\log\log n)$ amortized expected time update. At minimal update/ query time, this tradeoff gives $O(n^{4/3})$ space and $O(n^{2/3}\log n/\log\log n)$ worst-case time range mode query and amortized expected time update.

2.4 Lower Bounds

Both [3] and [4] give conditional lower bounds for range mode and range least frequent problems, respectively. Chan et al. [3] reduces multiplication of two $\sqrt{n}\times\sqrt{n}$ boolean matrices to n range mode queries in an array of size $O(n)$. This indicates that with current knowledge, preprocessing an $O(n)$-sized range mode query data structure and answering n range mode queries cannot be done in better than $O(n^{\omega/2})$ time, where ω is the constant in the exponent of the running time of matrix multiplication. With $\omega = 2.3727$ [17], this implies with current knowledge that a range mode data structure must either have preprocessing time at least $\Omega(n^{1.18635})$ or query time at least $\Omega(n^{0.181635})$. Since we may choose to update an array rather than initializing it, this lower bound also indicates that a dynamic range mode data structure must have update/ query time at least $\Omega(n^{\omega/2-1})$.

In [3], Chan et al. also give another conditional lower bound for dynamic range mode. They reduce the multiphase problem of Pătraşcu [16] to dynamic range mode. A reduction from 3-SUM (given n integers, find three that sum to zero) is given by Pătraşcu [16] to the multiphase problem. Based on the conjecture that the 3-SUM problem cannot be solved in $O(n^{2-\varepsilon})$ time for any positive constant ε, this chain of reductions implies a dynamic range mode data structure must have polynomial time query or update.

The reduction of $\sqrt{n}\times\sqrt{n}$ boolean matrix multiplication to n $O(n)$-sized range mode queries can be adopted to achieve the same conditional lower bound for the range least frequent problem [4]. Upon examination, the conditional lower bound also applies to k-frequency queries.

An unconditional lower bound also exists in the cell probe model for the range mode and k-frequency problem. Any range mode/ k-frequency data structure that uses S memory cells of w-bit words needs $\Omega(\frac{\log n}{\log(Sw/n)})$ time to answer a query [12].

3 Preliminaries

Before we discuss the technical details of our results, it will be helpful to develop some notation.

As in [3, 4], we will denote the subarray of A from index i to index j as $A[i : j]$ and use array notation $A[i]$ to denote the element of A at index i. We will assume zero-based indexing throughout this paper.

Furthermore, for range frequency data structures, the actual type that array A stores is irrelevant; we only care about how many times each element occurs. It will be useful to think of the identity of an element as a color. Therefore, we may say the color c occurs with frequency f in range $A[l : r]$. This is to distinguish that we are not referring to a particular index but rather the identity of multiple indices in the range.

For simplicity, we will assume the number of elements n is a perfect cube; however, the results discussed easily generalize for arbitrary n. All log's in this paper are assumed to be base 2.

In all our proofs, we analyze space cost in words, that is, $O(\log n)$ collections of bits.

4 Data Structure Setup

The idea of our data structure will be to break up array A into $O(n^{1/3})$ evenly-spaced endpoints, so that there are $O(n^{2/3})$ elements between each endpoint. We will use capital letters L and R when referring to particular endpoints. We will also occasionally refer to the elements between two consecutive endpoints as a segment; therefore, there are $O(n^{1/3})$ segments in A.

Each color that occurs in A will be split into the following two disjoint categories:

- **Frequent Colors**: Any color that appears more than $n^{1/3}$ times in A.
- **Infrequent Colors**: Any color that appears at most $n^{1/3}$ times in A.

Note that there can be at most $n/n^{1/3} = n^{2/3}$ frequent colors in A at any point in time.

Our data structure will need to use dynamic arrays as auxiliary data structures. These dynamic arrays will be a modification of the simple two-level version of the data structure described by Goodrich and Koss [11]. We have the following lemma regarding the performance of these dynamic arrays:

▶ **Lemma 1.** *There is a dynamic array data structure D that occupies $O(n)$ space and supports:*

1. *$D[i]$: Retrieve/ Set the element at rank i, in $O(1)$ time.*
2. *$Insert(i, x)$: Insert the element x at rank i, in $O(\sqrt{n})$ time.*
3. *$Delete(i, x)$: Delete the element x at rank i, in $O(\sqrt{n})$ time.*
4. *$Rank(ptr)$: Determine the rank of the element pointed to by ptr, in $O(1)$ time.*

The modified data structure will be discussed in the full version of this paper. If desired, the data structure can be replaced by a balanced binary search tree, at the cost of additional logarithmic factors in the query times of our data structure.

We can now describe the base set of auxiliary data structures used throughout the paper:

1. Arrays $B_{L,R}$, for all pairs of endpoints L, R, indexed from 0 to $n^{1/3}$, so that

$$B_{L,R}[i] := \text{The number of infrequent colors with frequency } i \text{ in } A[L : R].$$

2. Dynamic arrays D_c, for every color c, so that

   $D_c[i]$:= The index in A of the ith occurrence of color c.

3. An array E parallel to A so that

   $E[i]$:= A pointer to the location in memory of index i in dynamic array $D_{A[i]}$.

4. A binary search tree F of endpoints. At each endpoint R, we store

   $F[R]$:= A binary search tree on frequent colors, giving their frequency in $A[0:R]$.

In regards to $B_{L,R}$, we will sometimes refer to the set of elements between endpoints L and R as the span of L and R.

We now analyze the space complexity and construction time.

▶ **Lemma 2.** *The base data structures take $O(n)$ space.*

Proof. The arrays $B_{L,R}$ have size $O(n^{1/3})$ and there are $O(n^{2/3})$ of them, so in total these take $O(n)$ space. Every index of A is present in exactly one of the D_c arrays, and each dynamic array takes linear space. Therefore, in total all D_c arrays take $O(n)$ space. Array E has exactly the same size as A and thus takes $O(n)$ space. The binary search trees in each node of F have size equal to the number of frequent colors, which is at most $n^{2/3}$. Since there are $O(n^{1/3})$ endpoints and thus nodes of F, this structure takes $O(n)$ space. ◀

▶ **Lemma 3.** *The base data structures can be initialized in $O(n^{4/3})$ time.*

Proof. We can count the number of occurrences of each color in $O(n \log n)$ time and determine for each color whether it is frequent or infrequent. Let A' be the array A without any frequent colors. We can scan A' $n^{1/3}$ times, starting from each endpoint, to build the arrays $B_{L,R}$. This will be done as follows. For each scan, we maintain an array T so that $T[c]$ denotes the number of occurrences of color c found so far. We also maintain the array $B_{L,*}$ which is the array B with endpoint L and a variable right endpoint, that is maintained as elements are scanned. When $A[i] = c$, we check the number of occurrences of c to update $B_{L,*}$ to the correct state. In total, each element scanned results in $O(1)$ operations, until we reach a right endpoint. When we reach a right endpoint, we write $B_{L,*}$ to array $B_{L,R}$, where R denotes the right endpoint just encountered. In this way, for all endpoints L, R, we spend $O(n^{1/3})$ time to create the array. Thus the element scan dominates the time complexity, requiring $O(n^{4/3})$ time to create all $B_{L,R}$ arrays.

The dynamic arrays D_c can be built in linear time overall by walking through A and appending indices to the ends of D_c arrays. At this same time E can be built. The BSTs for all endpoints in F can also be built in linear time overall, since there are at most $n^{2/3}$ frequent colors per list, there are $n^{1/3}$ lists in total, and counting each frequent element in each interval can be done at the same time in one scan through A. ◀

Throughout the next few sections, we will use the following Lemma, as in [3]:

▶ **Lemma 4.** *Let $A[i] = c$. Then, given frequency f and right endpoint j, our base data structures may answer the following questions in constant time:*

Does color c occur at least f times in $A[i:j]$? (1)

Does color c occur at most f times in $A[i:j]$? (2)

Does color c occur exactly f times in $A[i:j]$? (3)

Proof. We call $rank(E[i])$ on D_c to get the rank r of i in D_c. Since array D_c stores the indices of every occurrence of color c in A, the values of $D_c[r+f]$ and $D_c[r+f-1]$ determine the answers to the above questions. ◀

A similar strategy can be used to answer the above questions given an index j and left endpoint i.

5 Range Mode Query

The range mode query will make use of the following lemma, originating from [14] and also used by [3]:

▶ **Lemma 5** (Krizanc et al. [14]). *Let A_1 and A_2 be any multisets. If c is a mode of $A_1 \cup A_2$ and $c \notin A_1$, then c is a mode of A_2.*

The query algorithm can be summarized as follows:

Algorithm 1 Range Mode Query in $A[l:r]$.

```
Let $L$ and $R$ be the first and last endpoints in $[l,r]$, respectively.
1. Check the frequency of every frequent color in $A[L:R]$ via BSTs $F[R]$ and $F[L]$. Let $f$
   be the highest frequency found so far.
2. Ask question (1) for all colors in $A[l:L-1] \cup A[R+1:r]$ with frequency $f$ and right
   endpoint $r$/ left endpoint $l$. If (1) is answered in the affirmative for color $c$, linearly scan
   $D_c$ to count the number of occurrences of color $c$ in $[l,r]$, update $f$, and continue.
3. Find the largest nonzero index of $B_{L,R}$. If this is larger than $f$, update $f$ and do the
   following:
   a. Find the next endpoint $R'$ to the left of $R$.
   b. Check $B_{L,R'}[f]$. If $B_{L,R'}[f] < B_{L,R}[f]$, search $A[R'+1:R]$ for a color that occurs $f$
      times in $A[L:R]$, via question (3) with left endpoint $L$.
   c. Otherwise, repeat from step (a) with $R \leftarrow R'$.
4. Return $f$ and the corresponding color found from either step 1, 2, or 3(b).
```

▶ **Theorem 6.** *Algorithm 1 finds the current range mode of $A[l:r]$ in $O(n^{2/3})$ time.*

Proof. Endpoints L and R can be found from $[l,r]$ by appropriate floors and ceilings in constant time.

In step 1, we can iterate through $F[R]$ and $F[L]$ in $O(n^{2/3})$ time, determining frequency counts for all frequent elements.

In step 2, answering question (1) takes $O(1)$ time per element via Lemma 4. Note that we use left endpoint l for elements in range $A[R+1:r]$ and right endpoint r for elements in range $A[l:L-1]$. Although we do not check (1) for the full range $[l,r]$, we will check the first/ last occurrence of any color in $A[l:L-1] \cup A[R+1:r]$, thereby effectively checking for all of $[l,r]$. When (1) is answered in the affirmative at index i, we linearly scan D_c, starting at $D_c[i+f+1]$ ($D_c[i-f-1]$ if $i \in [R+1:r]$) to determine a new highest frequency. Let us determine the cost of these linear scans. Let c be the most frequent color found from steps 1 and 2. If c is an infrequent color, it cannot occur more than $n^{1/3}$ times, so the total cost of the linear scans is no more than $O(n^{1/3})$. If it is a frequent color, its frequency cannot have increased by more than $O(n^{2/3})$, since its frequency was checked in step 1 and only $O(n^{2/3})$ elements exist in $A[l:L-1] \cup A[R+1:r]$. Thus the total cost of the linear scans is no more than $O(n^{2/3})$. In either case, step 2 takes $O(n^{2/3})$ time.

For step 3, finding the largest nonzero index of $B_{L,R}$ takes $O(n^{1/3})$ time. If this is larger than the frequencies found in steps 1 or 2, we execute steps 3(a) - 3(c). We can only repeat the steps at most $O(n^{1/3})$ times. The condition $B_{L,R'}[f] < B_{L,R}[f]$ will happen for one of these iterations, since eventually $R' = L$ in which case there are no elements accounted for in the interval. When $B_{L,R'}[f] < B_{L,R}[f]$, this implies a color with frequency f in range $[L, R]$ appears somewhere in $A[R'+1 : R]$. There are only $O(n^{2/3})$ elements in $A[R'+1 : R]$. Checking if color c occurs exactly f times in $A[L : R]$ can be done by asking question (3).

For the correctness of the value returned in step 4, note that the mode of $A[l : r]$ is either an element in $A[l : L-1] \cup A[R+1 : r]$ or the mode of $A[L : R]$, by Lemma 5. The frequency of all colors in $A[l : L-1] \cup A[R+1 : r]$ is checked. Further, the frequency of infrequent and frequent colors for interval $A[L : R]$ is also checked in steps 1 and 3, respectively. Therefore the color and frequency returned in step 4 must be the mode of $A[l : r]$. Putting it together, we see Algorithm 1 is correct and takes $O(n^{2/3})$ time. ◀

6 Update Operation

The update operation will require us to keep the base data structures up to date. Given update $A[i] \leftarrow c$, there are two similar procedures that must occur: adjusting the data structures for the removal of current color $A[i]$ and adjusting the data structures for the addition of color c at index i.

The following algorithm can be used for both procedures. We note that if either the color removed is the last occurrence of its type or the color added is a new color, the list D_c will have to also be constructed/ deleted and $B_{L,R}$ should be modified to only reflect colors present in A. For simplicity we omit these details from Algorithm 2.

Algorithm 2 Update Base Data Structures for Addition/ Removal of Color c at Index i.

```
1. If color $c$ is infrequent prior to this operation, count how many times $c$ occurs in each span via $D_c$, decrementing the corresponding index of each $B$ array.
2. Adjust $D_c$ by adding/ removing $i$. If this is an add operation, set $E[i]$ to the memory location of $i$ in $D_c$.
3. If color $c$ remains or becomes infrequent after step 2, again count how many times $c$ occurs in each span via $D_c$, incrementing the corresponding index of each $B$ array.
4. If color $c$ became infrequent in step 2, delete its entry in all nodes of $F$. If color $c$ became frequent after step 2, add its frequencies to $F$. If color $c$ remained frequent after step 2, increment the frequencies of all prefixes including index $i$ in $F$.
```

▶ **Theorem 7.** *Algorithm 2 updates the base data structures for addition/ removal of color c at index i in $O(n^{2/3})$ time.*

Proof. If c is an infrequent color prior to the update, step 1 removes its contribution to all B arrays. Counting the frequency of color c in each interval can be done via $O(n^{1/3})$ searches through D_c, each taking $O(n^{1/3})$ time. We first start at the beginning, finding its frequency for all right endpoints with left endpoint fixed, then move the left endpoint to the next endpoint and repeat, etc. Step 1 in total takes $O(n^{2/3})$ time.

Adding or removing i from D_c takes $O(\sqrt{n})$ time, as given in Lemma 1. Adjusting $E[i]$ can be done during this operation, so in total step 2 takes $O(\sqrt{n})$ time. The analysis of step 3 is identical to step 1. In step 4, adding, deleting, or modifying the entry of color c in all/

some nodes of F takes $O(n^{1/3} \log n)$ time, since each node stores the list of frequent colors and their frequency counts in a BST. If color c just became frequent, it only occurs $O(n^{1/3})$ times in A, and thus counting its frequency from the beginning to each endpoint can be done in $O(n^{1/3})$ time. In total, step 4 takes at most $O(n^{1/3} \log n)$ time.

After the completion of Algorithm 2, B arrays, dynamic array D_c, array E, and binary search tree F has been updated to reflect the current state of array A. Since all steps execute in no more than $O(n^{2/3})$ time, Algorithm 2 is correct and runs in $O(n^{2/3})$ time. ◀

7 Range Least Frequent Query, Allowing Zero

To answer range least frequent queries, we require the use of one more auxiliary data structure: an array $C_{L,R}$ for all pairs of endpoints L, R, indexed from 1 to $n^{1/3}$. At index $C_{L,R}[i]$ we store a list of all colors that occur i times in A such that the smallest span enclosing all occurrences is $[L, R]$.

Since each infrequent color is represented exactly once in the $C_{L,R}$ lists and there are $O(n^{2/3} \cdot n^{1/3}) = O(n)$ total indices present, this data structure takes linear space. It can also be initialized in linear time, and we can modify the update procedure of Algorithm 2 to update $C_{L,R}$ at the same time as $B_{L,R}$ for no additional time cost.

It is additionally worth noting that since the Range Least Frequent Query will require $\tilde{O}(n^{2/3})$ time, the use of dynamic arrays for data structures D_c is not necessary for this section. Instead, we can use binary search trees, augmented to support lookup by index in $O(\log n)$ time, or Dietz' data structure [6]. For the best time complexity, we will use augmented binary search trees to count occurrences of colors in a specific range and an augmented dynamic linear-space van Emde Boas tree to count occurrences of colors between endpoints. The van Emde Boas tree stores a single node for any endpoint R that a color appears in, which keeps the number of occurrences of that color from the beginning to endpoint R. The van Emde Boas tree can be updated in $O(n^{1/3} \log \log n)$ time upon insertion or removal and via predecessor/ successor queries, can support counting the number of occurrences of any color between any two endpoints in $O(\log \log n)$ time.

Algorithm 3 Range Least Frequent Query in $A[l : r]$, Allowing Zero.

```
Let $L$ and $R$ be the first and last endpoints in $[l, r]$, respectively.
1. Check the frequency of every frequent color in $A[l : r]$ via $D_c$. Let $f$ be the lowest
   frequency found so far.
2. Find the set of all infrequent colors that occur in $A[l : L-1] \cup A[R+1 : r]$; call it $U$.
   Count the frequencies of all colors of $U$ in range $A[l : r]$ and update $f$ with the lowest
   frequency found so far.
3. Compute $B'_{L,R}$, the array $B_{L,R}$ updated to erase the contribution of all colors in $U$.
4. If the smallest positive-valued index of $B'_{L,R}$ is less than $f$, update $f$ and check if any list
   $C_{L',R'}[f]$, $[L', R'] \subseteq [L, R]$, is non-empty. If so, return a color from the appropriate list.
   Otherwise, binary search from $R$ to the last endpoint of $A$ in the following way:
   a. Let $R'$ be a value in the middle of the search range. If $B'_{L,R'}[f] < B'_{L,R}[f]$, let $R'$ be
      the new upper bound; otherwise, continue the search in the half of the range after $R'$.
   b. When the search range is two consecutive endpoints, we search the range for a color
      that occurs $f$ times in $A[L : R]$ and return its identity.
   c. If the condition in a. is never satisfied, we must repeat a binary search on the other
      side, with an initial search range of the beginning of $A$ to $L$.
```

▶ **Theorem 8.** *Algorithm 3 finds the least frequent element of $A[l : r]$, allowing zero, in $O(n^{2/3} \log n \log \log n)$ time.*

Proof. We can find a list of frequent colors in any node of the F BST. Using an augmented binary search tree for D_c, step 1 can then be done in $O(n^{2/3} \log n)$ time. In step 2, since there are $O(n^{2/3})$ elements in $A[l : L-1] \cup A[R+1 : r]$, we can find color set U and complete step 2 similarly to step 1 in $O(n^{2/3} \log n)$ time. Step 3 requires counting the frequency of each color of U in range $A[L : R]$ and decrementing the corresponding index to make $B'_{L,R}$; thus it can also be done in $O(n^{2/3} \log \log n)$ time using the augmented van Emde Boas tree.

In step 4 we are looking for a least frequent element of $A[L : R]$ that does not occur in $A[l : L-1] \cup A[R+1 : r]$. All colors of $A[l : L-1] \cup A[R+1 : r]$ are represented in set U, found in step 3. We effectively erase the contribution of colors of U via computing $B'_{L,R}$; therefore, we can proceed as if colors of U do not exist in A. We will refer to A' as array A without any colors of U.

If the least frequent color in $A'[L : R]$ does not exist elsewhere in A, then its identity will be stored in a list $C_{L',R'}[f]$, $[L', R'] \subseteq [L, R]$. Note colors of U need not be special-cased for this lookup, since by appearing in $A[l : L-1] \cup A[R+1 : r]$, they will not be present in any of the searched lists. Otherwise, we know the least frequent color in $A'[L : R]$ must occur somewhere else in A'.

Now, amongst all colors in A', we know frequency f is minimal in range $A'[L : R]$. Therefore if we increase the range to $A'[L : R']$, the frequency of colors can only increase. For this property to hold, we must allow $f = 0$.

In each iteration, the smallest positive-valued index of $B'_{L,R'}$ will be f or greater and $B'_{L,R'}[f]$ will be no more than $B'_{L,R}[f]$. If it is less, we know one of the colors that occurred f times in $A'[L : R]$ now occurs more than f times in $A'[L : R']$. Therefore we may find it in the half of the search range before R'. If it is the same, we know none of the colors that occurred f times in $A'[L : R]$ appear in the half of the search range before R'. Either way we decrease the search range by a factor of 2. When the range represents two consecutive endpoints, we can search it for a color that occurs f times in $A[l : r]$ in $O(n^{2/3} \log n)$ time.

However, if R' is the end of the array and $B'_{L,R'}[f] = B'_{L,R}[f]$, then none of the colors that appear f times in $A'[L : R]$ appear to the right of R in A'. In this case, we can repeat the same binary search on the other side, decreasing a left endpoint L' and checking the same condition. Since the least frequent color in $A'[L : R]$ must occur elsewhere in A, as it was not present in any of the lists $C_{L',R'}[f]$, the search on this side must identify a color that appears f times in $A'[L : R]$.

The time complexity of step 4 can be analyzed as follows. Checking the lists $C_{L',R'}$ takes $O(n^{2/3})$ time, since there can be $O(n^{2/3})$ endpoints $[L', R'] \subseteq [L, R]$. The binary search is on endpoints, of which there are $O(n^{1/3})$. Thus, there are $O(\log(n^{1/3})) = O(\log n)$ iterations of the binary search, and in each iteration we must compute $B'_{L,R'}$ or $B'_{L',R}$. This computation takes $O(n^{2/3} \log \log n)$ time as in step 3. Therefore the binary search process takes $O(n^{2/3} \log n \log \log n)$ time. In total, step 4 takes $O(n^{2/3} \log n \log \log n)$ time.

For correctness, the least frequent element in $A[l : r]$ is either a frequent or infrequent color. If it is a frequent color, it is identified in step 1. If it is an infrequent color, we have two cases. Either the color occurs in $A[l : L-1] \cup A[R+1 : r]$, and thus set U, or it does not occur in set U. Step 2 accounts for all infrequent colors in set U. Steps 3 and 4 account for the last case. By the above, these steps find an infrequent color in $A[L : R]$ that does not occur in $A[l : L-1] \cup A[R+1 : r]$ in $O(n^{2/3} \log n \log \log n)$ time. Thus, Algorithm 3 is correct and finds the least frequent element of $A[l : r]$, allowing zero, in $O(n^{2/3} \log n \log \log n)$ time. ◀

8 Range k-Frequency Query

The previous two sections make use of a monotonicity property to find a color of given frequency in a range: for range mode, we know the frequency of the most frequent element can only decrease if the query range is decreased; furthermore, for range least frequent, we know the frequency of the least frequent element can only increase if the query range is increased. For this monotonicity condition to hold for least frequent elements, we must allow answering with an element of frequency zero. To force our answer to be an element that occurs in the query range, we must use an additional data structure that allows retrieval of colors by frequency in the $B_{L,R}$ arrays. To achieve $\tilde{O}(n)$ space, we cannot afford to store a list of colors at each index $B_{L,R}[i]$, and storing a single color at each index will run into issues during updates. Instead, we will use the following data structure which is similar to randomized data structures in the dynamic streaming literature [8]:

▶ **Lemma 9.** *There is a Monte Carlo data structure that occupies expected $O(\log^2 n)$ space and supports:*

1. *$Insert(x)$: Insert element x into the collection, in $O(\log n)$ expected time.*
2. *$Delete(x)$: Delete element x from the collection, in $O(\log n)$ expected time.*
3. *$Retrieve()$: Return an element in the collection, in $O(1)$ expected time.*

The data structure requires the $Delete(x)$ operation is executed independently of the results of $Retrieve()$.

The low-space data structure will be described in the full version of this paper. With it, we can answer the general problem of finding an element of given frequency in a query range. The additional auxilliary data structures needed will be as follows:

1. An array $G_{L,R}$ parallel to $B_{L,R}$. At index $G_{L,R}[i]$, we store the number of infrequent colors with frequency i in $A[L : R]$, excluding colors that appear in segments immediately left of L or right of R.
2. An array $H_{L,R}$, parallel to $G_{L,R}$, so that at index $H_{L,R}[i]$, we store a collection of colors counted in $G_{L,R}[i]$ in the data structure of Lemma 9.

The array $G_{L,R}$ is similar to the item (ii) stored in table D of [4]. During preprocessing, G arrays can be built similarly to B; however, when we fix left endpoint, we check all colors that occur in the segment to the left. We avoid counting such colors. Similarly, before we finalize the count for $G_{L,R}$, we move the right endpoint out as if to count the next range, keeping track of colors encountered. We subtract the frequency of such colors in $G_{L,R}$. Whenever we add to/ subtract from $G_{L,R}$, we can insert or delete the color from $H_{L,R}$.

As explained above, our space cost now becomes $\tilde{O}(n)$. Furthermore, our updates to G and H can be done alongside the update to B; however, since each insertion takes $O(\log n)$ time, the update procedure now takes $O(n^{2/3} \log n)$ time. With this we have:

Algorithm 4 Range k-Frequency Query in $A[l : r]$.

Let L and R be the first and last endpoints in $[l, r]$, respectively.

1. For color c at index i in the segment left of L, check via D_c to see if i is the first index of color c to appear in range $[l, r]$, or, if outside $[l, r]$, the next occurrence of color c lies in range $[l, r]$. If so, ask question (3) for the occurrence of color c in $[l, r]$ with frequency k and right endpoint r. If answered in the affirmative, return color c. Do the same, symmetrically, for colors in the segment right of R.
2. For each frequent color not addressed in step 1, check its frequency in $A[L : R]$ via BSTs $F[R]$ and $F[L]$. If any occur with frequency k, return the color.
3. If no color is found from step 1, check if $G_{L,R}[k] > 0$. If so, return $H_{L,R}[k].Retrieve()$. If not, return that no color has frequency k in range $A[l : r]$.

▶ **Theorem 10.** *Algorithm 4 returns an element of frequency k in $A[l : r]$ or indicates no such element exists, assuming update operations have been executed independently of results of query operations, in $O(n^{2/3})$ time.*

Proof. In step 1, we look at two full segments of $O(n^{2/3})$ total elements. For color c at index i, if $i \in [l, r]$, we must determine if index i is the first occurrence of color c in $[l, r]$. Let $m = D_c.rank(E[i])$. Index i is the first occurrence of color c in $[l, r]$ if $D_c[m - 1]$ is outside $[l, r]$. Similarly, if $i \notin [l, r]$, we can again define $m = D_c.rank(E[i])$, then check if $D_c[m + 1]$ is in $[l, r]$. In any case, for any color that occurs in segments immediately left of L or right of R, one of the indices will be the first outside $[l, r]$ or the first within $[l, r]$. Thus the frequency of the color will be checked in $A[l : r]$. Since we do a constant number of constant time operations for $O(n^{2/3})$ elements, step 1 takes $O(n^{2/3})$ time.

As in Algorithm 1, step 2 takes $O(n^{2/3})$ time. Step 3 takes $O(1)$ time. In any case, in step 1 we check all elements in segments immediately left of L or right of R to see if they occur k times in $A[l : r]$. Furthermore, in steps 2 and 3, we check every frequent and infrequent color that occurs in $A[L : R]$ but not in segments immediately left of L or right of R, via array G. Thus we have checked every color if it occurs k times in $A[l : r]$. Since no step takes more than $O(n^{2/3})$ time, this proves Theorem 10. ◀

Algorithm 4 can be easily modified to return the least frequent element present in the query range with the same time complexity and independence assumption. It can also be modified to count the number of elements above, below, or at a given frequency, as well as only determine the frequency of the least frequent element. Since these queries do not ask for a color, arrays $H_{L,R}$ are not needed. This reduces the space cost to $O(n)$, update cost to $O(n^{2/3})$, and requires no independence assumption.

References

1 Prosenjit Bose, Evangelos Kranakis, Pat Morin, and Yihui Tang. Approximate range mode and range median queries. In *Annual Symposium on Theoretical Aspects of Computer Science*, 2005.

2 Gerth Stølting Brodal, Beat Gfeller, Allan Grønlund Jørgensen, and Peter Sanders. Towards optimal range medians. *Theoretical Computer Science*, 412(24):2588–2601, 2011.

3 Timothy M. Chan, Stephane Durocher, Kasper Green Larsen, Jason Morrison, and Bryan T. Wilkinson. Linear-space data structures for range mode query in arrays. *Theory of Computing Systems*, 55:719–741, 2014.

4 Timothy M. Chan, Stephane Durocher, Matthew Skala, and Bryan T. Wilkinson. Linear-space data structures for range minority query in arrays. *Algorithmica*, 72:901–913, 2015.

5 Timothy M Chan and Konstantinos Tsakalidis. Dynamic orthogonal range searching on the ram, revisited. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 77. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

6 Paul F. Dietz. Optimal algorithms for list indexing and subset rank. In *Workshop on Data Structures and Algorithms*, pages 39–46, 1989.

7 Amr Elmasry, Meng He, J Ian Munro, and Patrick K Nicholson. Dynamic range majority data structures. In *International Symposium on Algorithms and Computation*, pages 150–159. Springer, 2011.

8 Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. *International Journal of Computational Geometry & Applications*, 18(1/2):3–28, 2008.

9 Harold N. Gabow, Jon Louis Bentley, and Robert E. Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 135–143, 1984.

10 Travis Gagie, Meng He, and Gonzalo Navarro. Compressed dynamic range majority data structures. In *Data Compression Conference (DCC), 2017*, pages 260–269. IEEE, 2017.

11 Michael T. Goodrich and John G. Koss II. Tiered vectors: Efficient dynamic arrays for rank-based sequences. In *Workshop on Data Structures and Algorithms*, 1999.

12 Mark Greve, Allan Grønlund Jørgensen, Kasper Dalgaard Larsen, and Jakob Truelsen. Cell probe lower bounds and approximations for range mode. In *International Colloquium on Automata, Languages, and Programming*, 2010.

13 Meng He, J. Ian Munro, and Patrick K. Nicholson. Dynamic range selection in linear space. In *International Symposium on Algorithms and Computation*, 2011.

14 Danny Krizanc, Pat Morin, and Michiel Smid. Range mode and range median queries on lists and trees. In *International Symposium on Algorithms and Computation.*, pages 517–526, 2003.

15 Mihai Pǎtraşcu and Emanuele Viola. Cell-probe lower bounds for succinct partial sums. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 117–122. Society for Industrial and Applied Mathematics, 2010.

16 Mihai Pǎtraşcu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the forty-second annual ACM symposium on Theory of computing*, pages 603–610, 2010.

17 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the fourty-fourth annual symposium on theory of computing*, 2012.

Online Makespan Scheduling with Job Migration on Uniform Machines

Matthias Englert
DIMAP and Department of Computer Science, University of Warwick, Coventry, UK
m.englert@warwick.ac.uk

David Mezlaf
Department of Computer Science, TU Dortmund, Dortmund, Germany
david.mezlaf@tu-dortmund.de

Matthias Westermann
Department of Computer Science, TU Dortmund, Dortmund, Germany
matthias.westermann@cs.tu-dortmund.de

Abstract

In the classic minimum makespan scheduling problem, we are given an input sequence of n jobs with sizes. A scheduling algorithm has to assign the jobs to m parallel machines. The objective is to minimize the makespan, which is the time it takes until all jobs are processed. In this paper, we consider online scheduling algorithms without preemption. However, we allow the online algorithm to reassign up to k jobs to different machines in the final assignment.

For m identical machines, Albers and Hellwig (Algorithmica, 2017) give tight bounds on the competitive ratio in this model. The precise ratio depends on, and increases with, m. It lies between $4/3$ and ≈ 1.4659. They show that $k = O(m)$ is sufficient to achieve this bound and no $k = o(n)$ can result in a better bound.

We study m uniform machines, i.e., machines with different speeds, and show that this setting is strictly harder. For sufficiently large m, there is a $\delta = \Theta(1)$ such that, for m machines with only two different machine speeds, no online algorithm can achieve a competitive ratio of less than $1.4659 + \delta$ with $k = o(n)$.

We present a new algorithm for the uniform machine setting. Depending on the speeds of the machines, our scheduling algorithm achieves a competitive ratio that lies between $4/3$ and ≈ 1.7992 with $k = O(m)$. We also show that $k = \Omega(m)$ is necessary to achieve a competitive ratio below 2.

Our algorithm is based on a subtle imbalance with respect to the completion times of the machines, complemented by a bicriteria approximation algorithm that minimizes the makespan and maximizes the average completion time for certain sets of machines.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases online algorithms, competitive analysis, minimum makespan scheduling, job migration

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.26

1 Introduction

In the classic minimum makespan scheduling problem, we are given an input sequence of n jobs with sizes. A scheduling algorithm has to assign the jobs to m parallel machines. The objective is to minimize the makespan, which is the time it takes until all jobs are processed. This problem is NP-hard in the strong sense [20]. In this paper, we consider online scheduling without preemption. An online algorithm does not have knowledge about the input sequence

© Matthias Englert, David Mezlaf, and Matthias Westermann;
licensed under Creative Commons License CC-BY

in advance. Instead, it gets to know the input sequence job by job without knowledge about the future. An online algorithm is called c-competitive if the makespan of the algorithm is at most c times the makespan of an optimal offline solution.

Extensive work has been done to narrow the gap between lower and upper bounds on the competitive ratio for online minimum makespan scheduling. Increasingly sophisticated algorithms and complex analyses were developed. Nevertheless, even for the most basic case of identical machines, in which each job has the same processing time, i.e., its size, on every machine, there is still a gap between the best known lower and upper bounds on the competitive ratio of 1.880 [30] and 1.9201 [18], respectively. In the setting with uniform machines, in which different machines may run at different speeds, the best known lower and upper bounds on the competitive ratio are 2.564 [13] and 5.828 [6], respectively.

In this work, we study to what extent the ability to migrate a limited number of jobs can help an online algorithm in terms of the competitive ratio in the uniform machine setting. In this model, the online algorithm has to assign jobs to machines as they arrive. However, after all jobs have arrived, the algorithm may remove up to k jobs from the machines and reassign them to different machines.

Job migration in scheduling has been studied previously, see for example [8, 12, 27, 32, 33, 34], but in particular, Albers and Hellwig [2] studied this problem for m identical machines[1] and gave tight bounds on the competitive ratio for this case. Roughly speaking, $k = \Theta(m)$ job migrations are sufficient and necessary to achieve this tight bound. Allowing more job migrations does not result in further improvements as long as $k = o(n)$, where n denotes the total number of arriving jobs.

We provide related results for the more general setting of uniform machines, which introduces new technical challenges. Our contribution also implies new results on a different but related problem: online reordering for scheduling. In this model, a so-called *reordering buffer* can be used to reorder the input sequence of jobs in a restricted fashion. Arriving jobs are first stored in the reordering buffer which has capacity to store up to k jobs. When the buffer is full, the online scheduling algorithm has to decide which of the jobs to remove from the buffer and to assign (irrevocably) to a machine. When no more jobs arrive, all jobs remaining in the buffer have to be assigned to machines as well.

This model was introduced by Englert, Özmen, and Westermann [14] and the work by Albers and Hellwig [2] generalizes their results for identical machines to the setting were no buffer is used, but a limited number of job migrations are permitted. It is not known what the relationship between the two models is in general. However, Albers and Hellwig note that any online algorithm for the job migration model that satisfies a certain monotonicity property can be transformed into an online algorithm for the corresponding reordering buffer problem which has the same competitive ratio. If the algorithm migrates k jobs, the transformed algorithm requires a buffer of size k. The aforementioned monotonicity property is as follows: if the algorithm would not migrate a job at time t if we pretend that the input sequence ends at that time, then the algorithm does not migrate the job at any later time either.

Both the algorithm by Albers and Hellwig and the algorithm we present in this work satisfy the monotonicity property. Therefore, our results also directly imply an improved upper bound for the online minimum makespan scheduling problem with a reordering buffer on uniform machines.

[1] Technically, they allow job migration to be performed before all jobs have arrived as long as the total number of migration is still bounded by k. However, performing all migrations at the end cannot increase the competitive ratio.

1.1 The model and our contribution

We present a lower bound on the competitive ratio showing that the problem is strictly harder for uniform machines than for identical machines. We give the first online algorithm for uniform machines with job migration. Depending on the speeds of the m machines, our scheduling algorithm achieves a competitive ratio that lies between $4/3$ and ≈ 1.7992 and performs $O(m)$ job migrations. In addition, we show that $\Omega(m)$ job migrations are necessary to achieve a competitive ratio of less than 2.

For the corresponding problem of online minimum makespan scheduling with a reordering buffer, Englert, Özmen, and Westermann [14] present a greedy algorithm that achieves a competitive ratio of 2 (or $2+\varepsilon$ if the algorithm is supposed to be efficient) with a reordering buffer of size m. Subsequently, Ding et al. [9] improved the competitive ratio to $2-1/m$ with a buffer of size $m+1$.[2] Therefore, we also obtain a significant improvement over these previously known results for the reordering buffer version of the problem, since our upper bound translates to this model as well.

Before we explain our contribution in more detail, we define the model more formally and introduce some useful notation and definitions. The $m \geq 2$ *machines* are denoted by $M_0, \dots, M_{m-1}$. For each $0 \leq i \leq m-1$, the *speed* of machine M_i is denoted by s_i, with $\min\{s_0, \dots, s_{m-1}\} = 1$. The *sum of speeds* is denoted by $S = \sum_{i=0}^{m-1} s_i$. The *size* of a job J is denoted by $p(J)$. The *load* $L(M_i)$ of a machine M_i is defined as the sum of the sizes of the jobs assigned to machine M_i. The *completion time* of a machine M_i is defined as the load $L(M_i)$ of machine M_i divided by the speed s_i of machine M_i. The objective is to minimize the makespan, i.e., the maximum completion time.

As in previous works of Englert, Özmen, and Westermann [14] and Albers and Hellwig [2], our algorithm attempts to maintain a specific (and not balanced) load distribution on the machines. The desired load on a machine M_i is defined by the so-called *weight* w_i of the machine. The weight is defined as

$$w_i = \begin{cases} s_i \cdot \frac{r_{s_0,\dots,s_{m-1}}}{S}, & \text{if } 0 \leq \sum_{j=0}^{i-1} s_j \leq \frac{r_{s_0,\dots,s_{m-1}}-1}{r_{s_0,\dots,s_{m-1}}} \cdot S \\ s_i \cdot \frac{r_{s_0,\dots,s_{m-1}}-1}{\sum_{j=0}^{i-1} s_j}, & \text{if } \frac{r_{s_0,\dots,s_{m-1}}-1}{r_{s_0,\dots,s_{m-1}}} \cdot S < \sum_{j=0}^{i-1} s_j < S \end{cases} .$$

Now, $r_{s_0,\dots,s_{m-1}}$ is the smallest positive solution to $\sum_{i=0}^{m-1} w_i = 1$, i.e., we ensure that the weights of all machines sum up to 1. Such a solution always exists. (Due to space limitations, the proof of this claim is omitted.) Note that, if $s_0 = \cdots = s_{m-1} = 1$, the weights match those in [2, 14] and that $r_{s_0,\dots,s_{m-1}} =: r_m$ is equal to the competitive ratio achieved in [2, 14] for m identical machines.

Unfortunately, we do not know a closed-form formula for $r_{s_0,\dots,s_{m-1}}$, but the value can be calculated for any given $s_0, \dots, s_{m-1}$ and $1 < r_{s_0,\dots,s_{m-1}} \leq \mathrm{W}_{-1}(-1/e^2)/(1+\mathrm{W}_{-1}(-1/e^2)) \approx 1.4659$.[3] (Due to space limitations, the proof of this claim is omitted.) Note that, for the optimal competitive ratio r_m for m identical machines, $4/3 \leq r_m \leq \mathrm{W}_{-1}(-1/e^2)/(1+\mathrm{W}_{-1}(-1/e^2))$. Depending on the speeds of the machines, $r_{s_0,\dots,s_{m-1}}$ can be significantly smaller than r_m.

[2] Note that in this and several of the following papers, the model differs from the model in [14] in that arriving jobs can bypass the buffer and may directly be assigned to a machine. This is equivalent to increasing the buffer size in the model from [14] by 1. We express buffer sizes in terms of the model from [14] here.

[3] W_{-1} is the lower branch of the Lambert W function, i.e., $\mathrm{W}_{-1}(-1/e^2)$ is the smallest real solution to $x \cdot e^x = -1/e^2$.

Our results are as follows.

- We prove that a $\delta = \Theta(1)$ exists such that, for m uniform machines with only two different machine speeds, m sufficiently large, no online algorithm can achieve a competitive ratio less than $\mathrm{W}_{-1}(-1/e^2)/(1+\mathrm{W}_{-1}(-1/e^2)) + \delta \approx 1.4659 + \delta$ while migrating $o(n)$ jobs. Recall that, for the optimal competitive ratio r_m for m identical machines, $r_m \leq \mathrm{W}_{-1}(-1/e^2)/(1+\mathrm{W}_{-1}(-1/e^2)) \approx 1.4659$. Hence, the more general problem of uniform machines is strictly harder than the special case of identical machines.

  The lower bound construction differs from the previous ones for identical machines in [2, 14]. The previous constructions used a very large number, say about $1/\varepsilon$, of very small jobs, say of size ε, which the online algorithm has to schedule on the machines. The adversary then identifies a machine with load of at least w_i, i.e., a machine with a load that is not below the "target load" and, roughly speaking, produces just enough large jobs so that one of them has to be assigned to a machine with load w_i. Migrating small jobs is ineffective and the large jobs cannot all avoid a machine with load w_i.

  This technique alone however is no longer sufficient to obtain a lower bound that is strictly larger than the known one. Using a larger number of possible continuations of the initial input, we can show that to handle these additional continuations, the online algorithm would have to have a significant number of machines with load strictly less than, and bounded away from, w_i. But then another machine must have load strictly above w_i (rather than just equal to w_i).

  We remark that the same lower bound can be constructed for the reordering buffer model with uniform machines.
- We show that, for m uniform machines, $\Omega(m)$ migrations are necessary to achieve a competitive ratio of less than 2. Specifically, for $c = \lceil -\ln(2-r)/\ln r \rceil \geq 2$ and $m \geq c^2$, no online algorithm can achieve a competitive ratio less than $r \in (1,2)$ while migrating at most $\lfloor m/c^2 \rfloor - 1$ jobs. For example, $r \approx 1.8393 > \mathrm{W}_{-1}(-1/e^2)/(1+\mathrm{W}_{-1}(-1/e^2)) + 1/3$ if at most $\lfloor m/9 \rfloor - 1$ job migrations are allowed.

  Again, we remark that the same lower bound can be constructed for the reordering buffer model with uniform machines.
- For m uniform machines with speeds $1 = s_0 \leq \cdots \leq s_{m-1}$, our online algorithm achieves a competitive ratio of $r_{s_0,\ldots,s_{m-1}} + 1/3$ with $O(m)$ job migrations. If an efficient algorithm is desired, there is an additional additive loss of ε in the competitive ratio due to the use of a PTAS by Hochbaum and Shmoys [24] in a subroutine. Note that $1 < r_{s_0,\ldots,s_{m-1}} \leq \mathrm{W}_{-1}(-1/e^2)/(1+\mathrm{W}_{-1}(-1/e^2)) \approx 1.4659$, i.e., the competitive ratio is at most an additive $1/3$ larger than in the identical machines case. However, depending on the speeds of the machines, $r_{s_0,\ldots,s_{m-1}}$ can also be significantly smaller than r_m in which case the difference between the competitive ratios can also be smaller than $1/3$.

  The basic structure of our algorithm is similar to the algorithm for the special case of identical machines [2]: Jobs are classified into small and large jobs according to their relative size compared to the total load on all machines. Ignoring the contribution of large jobs, the small jobs are scheduled in such a way that an imbalance with respect to the completion times of the machines is maintained. Roughly speaking, faster machines are kept at lower completion times than slower ones.

  After all jobs have arrived, some jobs are migrated. The rough intuition is that the largest jobs should be reassigned to improve the solution. For this, we first remove some jobs from machines. Then, we schedule the largest ones optimally on m empty virtual machines $M'_0, \ldots, M'_{m-1}$ with $L(M'_0) \leq \cdots \leq L(M'_{m-1})$. As a consequence, for each $0 \leq i \leq m-1$, the completion time of machine M'_i is less than or equal to the average completion time of the machines $M'_i, \ldots, M'_{m-1}$, which is a crucial property for achieving the optimal competitive ratio for identical machines. In the more general case of uniform machines, this is not always the case. For example, if M'_0 has speed 1 and $M'_1, \ldots, M'_{m-1}$ have speed 3/2, then m jobs of size 1 are optimally scheduled with makespan 1, but the completion time of M'_0 is 1, which is strictly greater than the average completion time of the machines $M'_0, \ldots, M'_{m-1}$.

To address this, our algorithm contains a crucial additional balancing step in which the average completion time for certain sets of virtual machines is increased at the cost of a small increase in the maximum completion time (which is responsible for the additive loss of 1/3).

Finally, the smaller jobs that were removed from their machines, are reassigned greedily one by one. The analysis of this step is also more involved than the corresponding one for identical machines because a more straightforward naive argument would introduce a factor of s_{m-1}/s_0 into the number of job migrations.

Obviously, once we determine which jobs to migrate, we could just assign those jobs optimally to the existing machines. However, it is not clear how to analyze such a procedure directly. We state a specific algorithm for the reassignment step because it provides us with important properties that enable us to analyze the competitive ratio.

1.2 Related work

Minimum makespan scheduling has been extensively studied. See the survey by Pruhs, Sgall, and Torng [29] for an overview. For m identical machines, the currently best upper and lower bounds are 1.9201 [18] and 1.880 [30], respectively. These bounds were the last ones in a long series of successive improvements for general or specific values of m [1, 4, 5, 7, 17, 21, 22, 25, 31].

For uniform machines, Aspnes et al. [3] present the first algorithm that achieves a constant competitive ratio. Due to Berman, Charikar and Karpinski [6], the best known upper bound on the competitive ratio is 5.828, and, due to Ebenlendr and Sgall [13], the best known lower bound on the competitive ratio is 2.564.

In a semi-online variant of the problem the jobs arrive in decreasing order of their size. The greedy LPT algorithm, which assigns each job to a machine with minimum load, was considered in this setting. For m identical machines, Graham [23] shows that the LPT algorithm achieves a competitive ratio of $4/3 - 1/(3m)$. For uniform machines, the LPT algorithm achieves a competitive ratio of 1.66 and a lower bound of 1.52 on its competitive ratio is known [19]. A detailed and tight analysis for two uniform machines is given by Mireault, Orlin, and Vohra [28] and Epstein and Favrholdt [15].

For m identical machines, Albers and Hellwig [2] present an algorithm that is r_m-competitive, which is optimal as long as at most $o(n)$ jobs can be migrated. For $m \geq 11$, the algorithm migrates at most $7m$ jobs. For smaller m, $8m$ to $10m$ jobs may be migrated. They further give some results on the trade-off between the number of job migrations and the competitive ratio. For example, $2.5 \cdot m$ job migrations are sufficient to achieve a competitive ratio of 1.75.

Tan and Yu [33] study two identical machines. They give a tight bound of 4/3 on the competitive ratio and this bound is achievable by migrating a single job. They also explore two other models. One in which, at the end, for each machine, the last job that was assigned to the machine may be migrated. And another in which, at the end, the k jobs that arrived last in the input may be migrated.

Chen et al. [8] give an optimal algorithm for two uniform machines. Using independent techniques and algorithms, Wang et al. [34] show bounds which are similar, but not quite optimal for all machine speeds. Both improve upon work by Liu et al. [27].

Dósa et al. [12] consider a variant in which up to k jobs can be migrated after every job arrival, which is a relaxation of online scheduling with a reordering buffer of size k. Sanders, Sivadasan, and Skutella [32] introduce another model in which, after every job arrival, a number of jobs can be reassigned as long as the total size of the reassigned jobs is bounded as some linear function of the size of the arriving job.

Numerous variants related to online minimum makespan scheduling with reordering buffers have been studied. Kellerer et al. [26] present, for two identical machines, an algorithm that achieves an optimal competitive ratio of $4/3$ with a reordering buffer of size 2, i.e., the smallest buffer size allowing reordering.

For m identical machines, Englert, Özmen, and Westermann [14] present a tight and, in comparison to the problem without reordering, improved bound on the competitive ratio for minimum makespan scheduling with reordering buffers. Depending on m, their scheduling algorithm achieves the optimal competitive ratio r_m with a buffer of size $\Theta(m)$. Further, they show that larger buffer sizes do not result in an additional advantage and that a buffer of size $\Omega(m)$ is necessary to achieve this competitive ratio.

Ding et al. [9] give, for m identical machines, a 1.5-competitive algorithm with a buffer of size $1.5m + 1$ and, for three identical machines, a (15/11)-competitive algorithm with a buffer of size 7.

Dósa and Epstein [10] study minimum makespan scheduling on two uniform machines with speed ratio $s \geq 1$. They show that, for any $s > 1$, a buffer of size 3 is sufficient to achieve an optimal competitive ratio and, in the case $s \geq 2$, a buffer of size 2 already allows to achieve an optimal ratio.

Dósa and Epstein [11] further study preemptive scheduling, as opposed to non-preemptive scheduling, on m identical machines with a reordering buffer. They present a tight bound on the competitive ratio for any m. This bound is $4/3$ for even values of m and slightly lower for odd values of m. They show that a buffer of size $\Theta(m)$ is sufficient to achieve this bound, but a buffer of size $o(m)$ does not reduce the best overall competitive ratio of $e/(e-1)$ that is known for the case without reordering.

Epstein, Levin, and van Stee [16] study the objective to maximize the minimum completion time. For m identical machines, they present an upper bound on the competitive ratio of $H_{m-1} + 1$ for a buffer of size m and a lower bound of H_m for any fixed buffer size. For m uniform machines, they show that a buffer of size $m + 2$ is sufficient to achieve the optimal competitive ratio m.

2 Lower bounds

Due to space limitations, the proofs of the following two theorems are omitted.

▶ **Theorem 1.** *A $\delta = \Theta(1)$ exists such that, for m uniform machines with only two machine speeds, m sufficiently large, no online algorithm can achieve a competitive ratio of less than $\mathrm{W}_{-1}(-1/e^2)/(1+\mathrm{W}_{-1}(-1/e^2)) + \delta \approx 1.4659 + \delta$ while migrating $o(n)$ jobs, where n denotes the total number of arriving jobs.*

▶ **Theorem 2.** *For $c = \lceil -\ln(2-r)/\ln r \rceil \geq 2$ and $m \geq c^2$ uniform machines, no online algorithm can achieve a competitive ratio of less than $r \in (1,2)$ while migrating at most $\lfloor m/c^2 \rfloor - 1$ jobs.*

3 Scheduling algorithm

For m uniform machines with speeds $1 = s_0 \leq \cdots \leq s_{m-1}$, our algorithm consists of two phases: In the scheduling phase, arriving jobs are assigned to (or scheduled on) machines online. In the migration phase, which starts after all jobs have arrived, some jobs are removed from their machines and reassigned to other machines.

More specifically, the scheduling phase consists of steps $1, \ldots, n$, where n denotes the total number of arriving jobs. In step t, the t-th job arrives and is assigned to a machine. For $t > 1$, let T_t denote the total size of the $t-1$ jobs that have arrived up to and including step $t-1$. In addition, define $T_1 = 0$. A job J is called *small in step* t, if $p(J) \leq T_t/(b \cdot m)$, where b is a constant that will be defined later. Otherwise, J is called *large in step* t. Note that during the scheduling phase, a job that is large in step t can become small in step $t+1$.

Further, let T_t^s denote the total size of the jobs that have arrived up to and including step $t-1$ and that are small in step t. Finally, let $L_t(M_i)$ denote the total size of the jobs that are scheduled on machine M_i at the end of step $t-1$, i.e., after the $(t-1)$-th job is assigned to a machine, and let $L_t^s(M_i)$ denote the total size of the jobs that are scheduled on machine M_i at the end of step $t-1$ and that are small in step t. For simplicity, define $r = r_{s_0,\ldots,s_{m-1}}$.

We use two different algorithms. The first algorithm, which is used when $s_{m-1} > 3/4 \cdot S$, schedules every job on machine M_{m-1} and does not migrate any jobs. The second algorithm, which is used when $s_{m-1} \leq 3/4 \cdot S$, is more interesting and works as follows.

- *Scheduling phase*: The t-th arriving job J is scheduled in step t as follows.
  - If J is small in step t, J is assigned to a machine M_i with $L_t^s(M_i) \leq w_i \cdot T_t^s$. (Since $\sum_{j=0}^{m-1} w_i = 1$ and $\sum_{i=0}^{m-1} L_t^s(M_i) = T_t^s$, such a machine always exists.)
  - If J is large in step t, J is assigned to a machine M_i that has minimum completion time $L_t(M_i)/s_i$ among all machines.
- *Migration phase*: Throughout the migration phase, we remove jobs from machines and reassign them. At any point during this process, let $L(M_i)$ denote the load of machine M_i at that point, i.e., the $L(M_i)$ values are dynamically changing throughout the migration phase.

  At the start of the migration phase, after all n jobs have arrived, we have, for each $0 \leq i \leq m-1$, $L(M_i) = L_{n+1}(M_i)$. Then do the following. For each machine M_i, as long as $L(M_i) > w_i \cdot T_{n+1}^s$ and $L(M_i) > (r-1) \cdot T_{n+1} \cdot s_i/S$, remove the job of largest size from M_i.

  The removed jobs can now be reassigned optimally to the machines, i.e., in such a way that the resulting makespan is minimized. However, as stated before, it is difficult to analyze the resulting makespan directly. In the following, we therefore present a more specific procedure for this reassignment step which provides us with certain properties that enable us to analyze the competitive ratio. The resulting bound is of course also an upper bound on the competitive ratio achieved through an optimal reassignment.

  (1) Those removed jobs that are large at time $n+1$ are scheduled on m empty virtual machines $M'_0, \ldots, M'_{m-1}$ with speeds $1 = s_0 \leq \cdots \leq s_{m-1}$:

  (1a) The jobs are scheduled on the virtual machines optimally, i.e., to minimize the makepsan of the virtual machines.[4] Call the resulting makespan on the virtual machines OPT$'$. We assume that the resulting loads of the virtual machines

[4] If computational efficiency is a concern, the PTAS by Hochbaum and Shmoys [24] may be used instead, resulting in an additive loss of ε in the competitive ratio.

are sorted, i.e., $L(M'_0) \leq \cdots \leq L(M'_{m-1})$, and that, for each $1 \leq i \leq m-1$, $L(M'_i)/s_i > \mathrm{OPT}'/2$ if $L(M'_{i-1}) > 0$. (See the following Observation 3 items (1) and (2).)

(1b) Each machine M'_i, with

$$i \in C = \left\{ 0 \leq i \leq m-1 : \sum_{j=0}^{m-1} L(M'_j) \leq \left(\frac{L(M'_i)}{s_i} - \frac{\mathrm{OPT}'}{3} \right) \cdot \sum_{j=i}^{m-1} s_j \right\} ,$$

is called *critical*. If $C \neq \emptyset$, all jobs from the machines $M'_0, \ldots, M'_c$, with $c = \max(C) < m-1$, are reassigned to the machines $M'_{c+1}, \ldots, M'_{m-1}$.

For $i = 0, \ldots, c$ do the following:

- Find the largest $\ell \geq c+1$ such that $(L(M'_i) + L(M'_\ell))/s_\ell \leq 4/3 \cdot \mathrm{OPT}'$. (Due to the following Observation 3 item (3), such a machine always exists.)
- Reassign all jobs from M'_i to M'_ℓ, i.e., $L(M'_\ell)$ is increased by $L(M'_i)$ and $L(M'_i)$ is set to 0.
- Resort the loads of the machines such that $L(M'_0) \leq \cdots \leq L(M'_{m-1})$ again. (See the following Observation 3 item (1).)

Finally, for each $0 \leq i \leq m-1$, assign the jobs from M'_i to the real machine M_i.

(2) Those removed jobs that are small at time $n+1$ are scheduled according to the greedy algorithm that assigns a job to a machine finishing it first.

Due to space limitations, the proof of the following observation is omitted.

▶ **Observation 3.** *For the migration phase, the following observations can be made.*

(1) *Sorting according to the load does not increase the makespan.*

(2) *We can assume that, for each* $1 \leq i \leq m-1$, $L(M'_i)/s_i > \mathrm{OPT}'/2$ *if* $L(M'_{i-1}) > 0$.

(3) *If* $C \neq \emptyset$, $\{c+1 \leq j \leq m-1 : (L(M'_i) + L(M'_j))/s_j \leq 4/3 \cdot \mathrm{OPT}'\} \neq \emptyset$.

(4) *For each* $0 \leq i \leq m-1$, $L(M'_i)/s_i \leq 4/3 \cdot \mathrm{OPT}'$.

3.1 Analysis of the algorithm

The analysis of the algorithm consists of two parts. The first part provides a bound on the number of migrated jobs. The second part provides a bound on the competitive ratio of the algorithm. These two parts together give the following theorem.

▶ **Theorem 4.** *For* m *uniform machines with speeds* $1 = s_0 \leq \cdots \leq s_{m-1}$*, our online algorithm achieves a competitive ratio of* $r_{s_0,\ldots,s_{m-1}} + 1/3$ *with* $O(m)$ *job migrations.*

3.1.1 Bounding the number of migrated jobs

The following lemma gives an upper bound on the number of jobs removed from a single machine.

▶ **Lemma 5.** *For each* $0 \leq i \leq m-1$*, in the migration phase, at most* $r/(r-1) \cdot b \cdot m \cdot s_i/S + 1$ *jobs are removed from machine* M_i.

Proof. If the final load of M_i at the end of the scheduling phase satisfies $L_{n+1}(M_i) \leq w_i \cdot T^s_{n+1}$ or $L_{n+1}(M_i) \leq (r-1) \cdot T_{n+1} \cdot s_i/S$, no job is removed from M_i. Otherwise, let t be the last time at which $L^s_t(M_i) \leq w_i \cdot T^s_{n+1}$ or $L_t(M_i) \leq (r-1) \cdot T_{n+1} \cdot s_i/S$. Such a time t exists because the condition is met for $t = 1$.

It is sufficient to remove the following jobs from M_i to guarantee $L(M_i) \leq w_i \cdot T^s_{n+1}$ or $L(M_i) \leq (r-1) \cdot T_{n+1} \cdot s_i/S$.

(a) All jobs that are large at time t and are scheduled on M_i before the arrival of the t-th job and
(b) all jobs assigned to M_i in step t or after.

At any time t' (before the arrival of the t'-th job), there are at most $b \cdot m \cdot s_i/S$ jobs that are large at time t' scheduled on M_i. Suppose this is not true and let t' be the first time at which this is not true. Then there were $b \cdot m \cdot s_i/S$ jobs of size greater than $T_{t'}/(b \cdot m)$ scheduled on M_i at time $t'-1$ and in step $t'-1$ one more such job J is assigned to M_i. However, before the assignment of J, the load of M_i is $L_{t'-1}(M_i) > T_{t'} \cdot s_i/S \geq T_{t'-1} \cdot s_i/S$. Then M_i cannot be a machine with minimum completion time among all machines in step t' and therefore a large job J would not be assigned to it. We conclude that, due to (a), at most $b \cdot m \cdot s_i/S$ jobs are removed.

To bound the number of jobs removed due to (b), we observe that in steps $t+1, \ldots, n$ our algorithm only allocates jobs to M_i that are large at the time of allocation. This is due to the fact that by definition of t, for each $t' \geq t+1$, $L^s_{t'}(M_i) > w_i \cdot T^s_{t'}$. Therefore, whenever a job J is assigned to M_i in a step $t' \geq t+1$, it is a large job, which is assigned to a machine of minimum completion time. But then, for each $0 \leq j \leq m-1$, $L_{t'}(M_j) > (r-1) \cdot T_{n+1} \cdot s_j/S$, because we also have $L_{t'}(M_i) > (r-1) \cdot T_{n+1} \cdot s_i/S$. Hence $T_{t'} = \sum_{j=0}^{m-1} L_{t'}(M_j) > (r-1) \cdot T_{n+1}$. Since job J is large at the time of assignment, its size has to be greater than $(r-1) \cdot T_{n+1}/(b \cdot m)$. After assigning $b \cdot m \cdot s_i/(S \cdot (r-1))$ such jobs to M_i in steps after t, the load of M_i exceeds $T_{n+1} \cdot s_i/S$. After that, no further such jobs are assigned to M_i, because a machine with load greater than $T_{n+1} \cdot s_i/S$ can never be a machine that has the smallest completion time among all machines. We conclude that, due to (b), at most $b \cdot m \cdot s_i/(S \cdot (r-1)) + 1$ jobs are removed, where the additive 1 is due to the job that is assigned to machine M_i in step t.

In total, it is sufficient to remove these $b \cdot m \cdot s_i/S + b \cdot m \cdot s_i/(S \cdot (r-1)) + 1 = r/(r-1) \cdot b \cdot m \cdot s_i/S + 1$ many jobs, and, because the algorithm removes jobs from M_i in decreasing order of size, the number of jobs removed is bounded by the same number. ◀

Recall, that we only migrate jobs when $s_{m-1} \leq 3/4 \cdot S$, as otherwise, we simply schedule all jobs on machine M_{m-1}. If $s_{m-1} \leq 3/4 \cdot S$, $18/17 \leq r \leq \mathrm{W}_{-1}(-1/e^2)/(1+\mathrm{W}_{-1}(-1/e^2)) \approx 1.4659$. (Due to space limitations, the proof of this claim is omitted.) Hence, due to Lemma 5, the total number of jobs migrated is bounded by

$$\sum_{i=0}^{m-1} \left(\frac{r}{r-1} \cdot b \cdot m \cdot \frac{s_i}{S} + 1 \right) = \left(\frac{r}{r-1} \cdot b + 1 \right) \cdot m = \Theta(m) \ .$$

3.1.2 Bounding the competitive ratio

If $s_{m-1} > 3/4 \cdot S$, we assign all jobs to machine M_{m-1}. The resulting makespan is $L_{n+1}(M_{m-1})/s_{m-1} = T_{n+1}/s_{m-1} < 4/3 \cdot T_{n+1}/S \leq 4/3 \cdot \mathrm{OPT}$, where OPT denotes the optimal makespan. Hence the competitive ratio is bounded by $1 + 1/3$.

For the reminder of the paper, we consider the case $s_{m-1} \leq 3/4 \cdot S$. The following lemma shows that, at the end of step (1b), there are no critical machines. In fact, it gives a lower bound on $\sum_{j=0}^{m-1} L(M'_j)$.

▶ **Lemma 6.** *At the end of step (1b), for each $0 \leq j \leq m-1$,*

$$\sum_{k=0}^{m-1} L(M'_k) \geq \left(\frac{L(M'_j)}{s_j} - \frac{\mathrm{OPT}'}{3} \right) \cdot \sum_{k=j}^{m-1} s_k \geq \left(\frac{L(M'_j)}{s_j} - \frac{\mathrm{OPT}}{3} \right) \cdot \sum_{k=j}^{m-1} s_k \ .$$

Proof. The second inequality is true because $\mathrm{OPT}' \le \mathrm{OPT}$ (optimally scheduling a subset of all jobs can only result in a smaller makespan than optimally scheduling all jobs). In the following, we prove the first inequality. If $C = \emptyset$, the lemma is true by definition of C. In the following, we consider the case $C \neq \emptyset$. At the end of step (1b), for each $0 \le j \le c$, $L(M_i') = 0$ and, as a consequence, the lemma is true for these machines. In the following, we show that the lemma is true for $M_{c+1}', \dots, M_{m-1}'$ after each reassignment in step (1b), if it is true for these machines before this reassignment.

Initially, at the beginning of step (1b), for each $c+1 \le j \le m-1$, M_j' is not critical by definition of c, i.e., the lemma is true for M_j'.

Now, consider a reassignment in step (1b). For each $0 \le j \le m-1$, let $L(M_i')$ and $\hat{L}(M_i')$ denote the load of machine M_i' before and after this reassignment, respectively. Assume that the lemma is true for $M_{c+1}', \dots, M_{m-1}'$ before this reassignment.

In this reassignment, all jobs from M_i', with $0 \le i \le c$, are reassigned to M_ℓ', with

$$\ell = \max\left\{c+1 \le j \le m-1 : \frac{L(M_i') + L(M_j')}{s_j} \le \frac{4}{3} \cdot \mathrm{OPT}'\right\} .$$

Then, resort the loads of the machines again. In detail,

$$z = \max\left\{\ell \le j \le m-1 : L(M_j') < L(M_i') + L(M_\ell')\right\} ,$$

i.e., after resorting, $\hat{L}(M_z') = L(M_i') + L(M_\ell')$, and, for each $\ell \le j \le z-1$, $\hat{L}(M_j') = L(M_{j+1}')$. In addition, for each $j \in \{c+1, \dots, m-1\} \setminus \{\ell, \dots, z\}$, $\hat{L}(M_j') = L(M_j')$. Note that, for each $c+1 \le j \le m-1$, $L(M_j') \le \hat{L}(M_j')$ and, if $j+1 \le m-1$, $\hat{L}(M_j') \le L(M_{j+1}')$.

It remains to show that the lemma is true for $M_\ell', \dots, M_z'$. Consider machine M_x' with $\ell \le x \le z$. If $\hat{L}(M_x')/s_x \le \mathrm{OPT}'$, then

$$\sum_{j=0}^{m-1} \hat{L}(M_j') \ge \sum_{j=x}^{m-1} \hat{L}(M_j') \ge \frac{\hat{L}(M_x')}{s_x} \cdot s_x + \sum_{j=x+1}^{m-1} \frac{2}{3} \cdot \mathrm{OPT}' \cdot s_j \ge \left(\frac{\hat{L}(M_x')}{s_x} - \frac{\mathrm{OPT}'}{3}\right) \cdot \sum_{j=x}^{m-1} s_j ,$$

since, by definition of ℓ, for each $\ell+1 \le j \le m-1$, $\hat{L}(M_j')/s_j \ge L(M_j')/s_j \ge (L(M_i') + L(M_j'))/(2s_j) > 2/3 \cdot \mathrm{OPT}'$.

In the following, we consider the case $\hat{L}(M_x')/s_x > \mathrm{OPT}'$. Due to space limitations, the proof of the following observation is omitted.

▶ **Observation 7.** *For each $x+1 \le j \le m-1$, $L(M_j')/s_j \ge 4/5 \cdot \mathrm{OPT}'$.*

Due to the fact that M_c' is critical,

$$\sum_{j=c}^{m-1} L(M_j') \le \sum_{j=0}^{m-1} L(M_j') \le \left(\frac{L(M_c')}{s_c} - \frac{\mathrm{OPT}'}{3}\right) \cdot \sum_{j=c}^{m-1} s_j .$$

As a consequence,

$$\sum_{j=c+1}^{m-1} L(M_j') \le \left(\frac{L(M_c')}{s_c} - \frac{\mathrm{OPT}'}{3}\right) \cdot \sum_{j=c+1}^{m-1} s_j \le \frac{2}{3} \cdot \mathrm{OPT}' \cdot \sum_{j=c+1}^{m-1} s_j ,$$

since $L(M_c')/s_c - \mathrm{OPT}'/3 \le L(M_c')/s_c \le \mathrm{OPT}'$.

Due to Observation 3 item (2), $\sum_{j=c+1}^{x} L(M_j') \ge 1/2 \cdot \mathrm{OPT}' \cdot \sum_{j=c+1}^{x} s_j$ and, due to Observation 7, $\sum_{j=x+1}^{m-1} L(M_j') \ge 4/5 \cdot \mathrm{OPT}' \cdot \sum_{j=x+1}^{m-1} s_j$. Hence,

$$\frac{2}{3} \cdot \mathrm{OPT}' \cdot \left(\sum_{j=c+1}^{x} s_j + \sum_{j=x+1}^{m-1} s_j\right) \ge \sum_{j=c+1}^{m-1} L(M_j') \ge \frac{1}{2} \cdot \mathrm{OPT}' \cdot \sum_{j=c+1}^{x} s_j + \frac{4}{5} \cdot \mathrm{OPT}' \cdot \sum_{j=x+1}^{m-1} s_j ,$$

i.e., $\sum_{j=c+1}^{x} s_j \geq 4/5 \cdot \sum_{j=x+1}^{m-1} s_j$.

Altogether,

$$\begin{aligned}
\sum_{j=0}^{m-1} \hat{L}(M'_j) &\geq \sum_{j=c+1}^{x-1} L(M'_j) + \hat{L}(M'_x) + \sum_{j=x+1}^{m-1} L(M'_j) \\
&\geq \frac{1}{2} \cdot \mathrm{OPT}' \cdot \sum_{j=c+1}^{x-1} s_j + \frac{1}{3} \cdot \mathrm{OPT}' \cdot s_x + \left(\frac{\hat{L}(M'_x)}{s_x} - \frac{\mathrm{OPT}'}{3} \right) \cdot s_x \\
&\quad + \frac{4}{5} \cdot \mathrm{OPT}' \cdot \sum_{j=x+1}^{m-1} s_j \\
&\geq \frac{1}{3} \cdot \mathrm{OPT}' \cdot \sum_{j=c+1}^{x} s_j + \left(\frac{\hat{L}(M'_x)}{s_x} - \frac{\mathrm{OPT}'}{3} \right) \cdot s_x + \frac{4}{5} \cdot \mathrm{OPT}' \cdot \sum_{j=x+1}^{m-1} s_j \\
&\geq \left(\frac{\hat{L}(M'_x)}{s_x} - \frac{\mathrm{OPT}'}{3} \right) \cdot s_x + \left(\frac{1}{3} \cdot \frac{4}{5} + \frac{4}{5} \right) \cdot \mathrm{OPT}' \cdot \sum_{j=x+1}^{m-1} s_j \\
&\geq \left(\frac{\hat{L}(M'_x)}{s_x} - \frac{\mathrm{OPT}'}{3} \right) \cdot \sum_{j=x}^{m-1} s_j \ ,
\end{aligned}$$

since $\hat{L}(M'_x)/s_x \leq 4/3 \cdot \mathrm{OPT}'$ due to Observation 3 item (4). ◀

Next, we give a bound on the makespan at the end of step (1) of the migration phase. We distinguish two cases.

- $L(M_i) \leq (r-1) \cdot T_{n+1} \cdot s_i/S$ after the removal of jobs:
  Then, $L(M_i) \leq (r-1) \cdot T_{n+1} \cdot s_i/S \leq (r-1) \cdot \mathrm{OPT} \cdot s_i$. The completion time of machine M_i at the end of step (1) of the migration phase is $(L(M_i) + L(M'_i))/s_i \leq (r-1) \cdot \mathrm{OPT} + 4/3 \cdot \mathrm{OPT} \leq (r + 1/3) \cdot \mathrm{OPT}$, since $L(M'_i)/s_i \leq 4/3 \cdot \mathrm{OPT}' \leq 4/3 \cdot \mathrm{OPT}$ due to Observation 3 item (4).
- $L(M_i) > (r-1) \cdot T_{n+1} \cdot s_i/S$ after the removal of jobs:
  Then, $L(M_i) \leq w_i \cdot T^s_{n+1}$ after the removal of jobs. We distinguish two sub-cases.
  - $w_i = s_i \cdot r/S$:
    By definition of w_i, $\sum_{j=0}^{i-1} s_j \leq (r-1)/r \cdot S$ and, as a consequence,

$$\sum_{j=i}^{m-1} s_j = S - \sum_{j=0}^{i-1} s_j \geq S - \frac{r-1}{r} \cdot S = \frac{S}{r} \ .$$

    Then we can bound the completion time of machine M_i at the end of step (1) of the migration phase as follows:

$$\begin{aligned}
\frac{L(M_i)}{s_i} &\leq \frac{w_i}{s_i} \cdot \left(S \cdot \mathrm{OPT} - \sum_{j=0}^{m-1} L(M'_j) \right) + \frac{L(M'_i)}{s_i} \\
&\leq \frac{r}{S} \cdot \left(S \cdot \mathrm{OPT} - \max\left\{ 0, \frac{L(M'_i)}{s_i} - \frac{\mathrm{OPT}}{3} \right\} \cdot \sum_{j=i}^{m-1} s_j \right) + \frac{L(M'_i)}{s_i} \\
&\leq \frac{r}{S} \cdot \left(S \cdot \mathrm{OPT} - \max\left\{ 0, \frac{L(M'_i)}{s_i} - \frac{\mathrm{OPT}}{3} \right\} \cdot \frac{S}{r} \right) + \frac{L(M'_i)}{s_i} \\
&\leq r \cdot \mathrm{OPT} + \frac{1}{3} \cdot \mathrm{OPT} \ .
\end{aligned}$$

- $w_i = s_i \cdot (r-1)/\sum_{j=0}^{i-1} s_j$:
  By definition of w_i,

$$\frac{r-1}{r} \cdot S \leq \sum_{j=0}^{i-1} s_j \ .$$

Then we can bound the completion time of machine M_i at the end of step (1) of the migration phase as follows:

$$\begin{aligned}
\frac{L(M_i)}{s_i} &\leq \frac{w_i}{s_i} \cdot \left(S \cdot \mathrm{OPT} - \sum_{j=0}^{m-1} L(M'_j) \right) + \frac{L(M'_i)}{s_i} \\
&\leq \frac{r-1}{\sum_{j=0}^{i-1} s_j} \cdot \left(S \cdot \mathrm{OPT} - \max\left\{0, \frac{L(M'_i)}{s_i} - \frac{\mathrm{OPT}}{3}\right\} \cdot \left(S - \sum_{j=0}^{i-1} s_j \right) \right) + \frac{L(M'_i)}{s_i} \\
&\leq \frac{r-1}{\sum_{j=0}^{i-1} s_j} \cdot S \cdot \left(\mathrm{OPT} - \max\left\{0, \frac{L(M'_i)}{s_i} - \frac{\mathrm{OPT}}{3}\right\} \right) \\
&\quad + (r-1) \cdot \max\left\{0, \frac{L(M'_i)}{s_i} - \frac{\mathrm{OPT}}{3}\right\} + \frac{L(M'_i)}{s_i} \\
&\leq r \cdot \mathrm{OPT} - \max\left\{0, \frac{L(M'_i)}{s_i} - \frac{\mathrm{OPT}}{3}\right\} + \frac{L(M'_i)}{s_i} \\
&\leq r \cdot \mathrm{OPT} + \frac{1}{3} \cdot \mathrm{OPT} \ ,
\end{aligned}$$

since $\sum_{j=i}^{m-1} s_j = S - \sum_{j=0}^{i-1} s_j$ and $4/3 \cdot \mathrm{OPT} - L(M'_i)/s_i \geq 4/3 \cdot \mathrm{OPT}' - L(M'_i)/s_i \geq 0$ due to Observation 3 item (4).

In all cases, the makespan is at most $(r + 1/3) \cdot \mathrm{OPT}$ at the end of step (1) of the migration phase.

Finally, we analyze the makespan at the end of step (2) of the migration phase. We start with the following observation. Due to space limitations, the proof of this observation is omitted.

▶ **Observation 8.** *There exists a machine M_i with $m_b + 1 \leq i \leq m-1$ and completion time of at most $(\sqrt{b}+1)/\sqrt{b} \cdot \mathrm{OPT}$, where*

$$m_b = \max\left\{ 0 \leq i \leq m-1 : \sum_{j=0}^{i} s_j < \frac{S}{\sqrt{b}+1} \right\} < m-1 \ .$$

Consider a removed job J that is scheduled in step (2) of the migration phase. Since J is small at time $n+1$, $p(J) \leq T_{n+1}/(b \cdot m) \leq \mathrm{OPT} \cdot S/(b \cdot m)$. According to Observation 8, there exists a machine M_i with $m_b + 1 \leq i \leq m-1$ and completion time of at most $(\sqrt{b}+1)/\sqrt{b} \cdot \mathrm{OPT}$. Since $\sum_{j=0}^{m_b+1} s_j \geq S/(\sqrt{b}+1)$, $s_i \geq \sum_{j=0}^{i} s_j/(i+1) \geq S/((\sqrt{b}+1) \cdot m)$. In step (2) of the migration phase, J is assigned to a machine finishing it first. Then, we can bound the completion time of this machine after J is assigned to it as follows:

$$\frac{L(M_i)}{s_i} + \frac{p(J)}{s_i} \leq \frac{\sqrt{b}+1}{\sqrt{b}} \cdot \mathrm{OPT} + \mathrm{OPT} \cdot \frac{S}{b \cdot m} \cdot \frac{(\sqrt{b}+1) \cdot m}{S} = \left(\frac{\sqrt{b}+1}{\sqrt{b}} \right)^2 \cdot \mathrm{OPT} \ .$$

At the end of the migration phase, the makespan is at most $\max\{r+1/3, (1+1/\sqrt{b})^2\} \cdot \mathrm{OPT}$. Recall that $1 < r \leq \mathrm{W}_{-1}(-1/e^2)/(1 + \mathrm{W}_{-1}(-1/e^2)) \approx 1.4659$. For example, for $b = 8.5827$, $(1 + 1/\sqrt{b})^2 \leq 1.4659 + 1/3$, and, for $b = 41.7847$, $(1 + 1/\sqrt{b})^2 \leq 4/3$.

References

1 Susanne Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459–473, 1999.

2 Susanne Albers and Matthias Hellwig. On the value of job migration in online makespan minimization. *Algorithmica*, 79(2):598–623, 2017.

3 James Aspnes, Yossi Azar, Amos Fiat, Serge A. Plotkin, and Orli Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44(3):486–504, 1997.

4 Yair Bartal, Amos Fiat, Howard J. Karloff, and Rakesh Vohra. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, 51(3):359–366, 1995.

5 Yair Bartal, Howard J. Karloff, and Yuval Rabani. A better lower bound for on-line scheduling. *Information Processing Letters*, 50(3):113–116, 1994.

6 Piotr Berman, Moses Charikar, and Marek Karpinski. On-line load balancing for related machines. *Journal of Algorithms*, 35(1):108–121, 2000.

7 Bo Chen, André van Vliet, and Gerhard J. Woeginger. New lower and upper bounds for on-line scheduling. *Operations Research Letters*, 16(4):221–230, 1994.

8 Xin Chen, Yan Lan, Attila Benko, György Dósa, and Xin Han. Optimal algorithms for online scheduling with bounded rearrangement at the end. *Theoretical Computer Science*, 412(45):6269–6278, 2011.

9 Ning Ding, Yan Lan, Xin Chen, György Dósa, He Guo, and Xin Han. Online minimum makespan scheduling with a buffer. *International Journal of Foundations of Computer Science*, 25(5):525–536, 2014.

10 György Dósa and Leah Epstein. Online scheduling with a buffer on related machines. *Journal of Combinatorial Optimization*, 20(2):161–179, 2010.

11 György Dósa and Leah Epstein. Preemptive online scheduling with reordering. *SIAM Journal on Discrete Mathematics*, 25(1):21–49, 2011.

12 György Dósa, Yuxin Wang, Xin Han, and He Guo. Online scheduling with rearrangement on two related machines. *Theoretical Computer Science*, 412(8-10):642–653, 2011.

13 Tomás Ebenlendr and Jirí Sgall. A lower bound on deterministic online algorithms for scheduling on related machines without preemption. *Theory of Computing Systems*, 56(1):73–81, 2015.

14 Matthias Englert, Deniz Özmen, and Matthias Westermann. The power of reordering for online minimum makespan scheduling. *SIAM Journal on Computing*, 43(3):1220–1237, 2014.

15 Leah Epstein and Lene M. Favrholdt. Optimal preemptive semi-online scheduling to minimize makespan on two related machines. *Operations Research Letters*, 30(4):269–275, 2002.

16 Leah Epstein, Asaf Levin, and Rob van Stee. Max-min online allocations with a reordering buffer. *SIAM Journal on Discrete Mathematics*, 25(3):1230–1250, 2011.

17 Ulrich Faigle, Walter Kern, and György Turán. On the performance of on-line algorithms for partition problems. *Acta Cybernetica*, 9(2):107–119, 1989.

18 Rudolph Fleischer and Michaela Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343–353, 2000.

19 Donald K. Friesen. Tighter bounds for LPT scheduling on uniform processors. *SIAM Journal on Computing*, 16(3):554–560, 1987.

20 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

21 Todd Gormley, Nick Reingold, Eric Torng, and Jeffery Westbrook. Generating adversaries for request-answer games. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 564–565, 2000.

22 Ronald L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(1):1563–1581, 1966.

23 Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.

24 Dorit S. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.

25 David R. Karger, Steven J. Phillips, and Eric Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20(2):400–430, 1996.

26 Hans Kellerer, Vladimir Kotov, Maria Grazia Speranza, and Zsolt Tuza. Semi on-line algorithms for the partition problem. *Operations Research Letters*, 21(5):235–242, 1997.

27 Ming Liu, Yinfeng Xu, Chengbin Chu, and Feifeng Zheng. Online scheduling on two uniform machines to minimize the makespan. *Theoretical Computer Science*, 410(21-23):2099–2109, 2009.

28 Paul Mireault, James B. Orlin, and Rakesh V. Vohra. A parametric worst case analysis of the LPT heuristic for two uniform machines. *Operations Research*, 45(1):116–125, 1997.

29 Kirk Pruhs, Jiri Sgall, and Eric Torng. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter Online Scheduling. CRC Press, 2004.

30 John F. Rudin III. *Improved Bound for the Online Scheduling Problem.* PhD thesis, University of Texas at Dallas, 2001.

31 John F. Rudin III and R. Chandrasekaran. Improved bound for the online scheduling problem. *SIAM Journal on Computing*, 32(3):717–735, 2003.

32 Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2):481–498, 2009.

33 Zhiyi Tan and Shaohua Yu. Online scheduling with reassignment. *Operations Research Letters*, 36(2):250–254, 2008.

34 Yuxin Wang, Attila Benko, Xin Chen, György Dósa, He Guo, Xin Han, and Cecilia Sik-Lányi. Online scheduling with one rearrangement at the end: Revisited. *Information Processing Letters*, 112(16):641–645, 2012.

Truthful Prompt Scheduling for Minimizing Sum of Completion Times

Alon Eden
Tel Aviv University, Israel
alonarden@gmail.com

Michal Feldman
Tel Aviv University and Microsoft Research, Israel
michal.feldman@cs.tau.ac.il

Amos Fiat
Tel Aviv University, Israel
fiat@tau.ac.il

Tzahi Taub
Tel Aviv University, Israel
tzahita@gmail.com

Abstract

We give a prompt online mechanism for minimizing the sum of [weighted] completion times. This is the first prompt online algorithm for the problem. When such jobs are strategic agents, delaying scheduling decisions makes little sense. Moreover, the mechanism has a particularly simple form of an anonymous menu of options.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms, Theory of computation → Algorithmic mechanism design

Keywords and phrases Scheduling, Mechanism design, Online algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.27

Related Version A full version of the paper is available at [6], `https://arxiv.org/abs/1804.03244`.

Funding This work was partially supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement number 337122, by the ISF (grant number 317/17), and by the ISF (grant number 1841/14).

1 Introduction

The setting herein includes [multiple] service queues and selfish agents that arrive online over time and can be processed on one of m machines. Agents may have some (private) *processing time* p and/or some private *weight* w.

The goal is to improve service as much as possible. Minimizing the sum of [weighted] completion times is one measure of how good (or bad) service really is.

This problem has long been studied, as a pure optimization problem, without strategic considerations [11]. Given a collection of jobs, processing times, and weights, the shortest weighted processing time order [26], also known as Smith's rule, produces a minimal sum of weighted completion times with a non-preemptive schedule on a single machine.

© Alon Eden, Michal Feldman, Amos Fiat, and Tzahi Taub;
licensed under Creative Commons License CC-BY

Schedules can be preemptive (where jobs may be stopped and restarted over time) or non-preemptive (where a job, once execution starts, cannot be stopped until the job is done).

To the best of our knowledge, all online algorithms for this problem have the following property: when a job arrives, there are no guarantees as to when it will finish. If preemption is allowed, even if the job starts, there is no guarantee that it will not be preempted, or for how long. If preemption is disallowed, the online algorithm keeps the job "hanging about" for some unknown length of time, until the algorithm finally decides that it is time to start it.

Essentially, this means that when one requests service, the answer is "OK – just hang around and you will get service at some unknown future date". It is in fact impossible to achieve any bounded ratio for the sum of [weighted] completion times if one has to start processing the job as soon as possible[1]. Some delay is inevitable. However, the issue we address is "does the job know when it will be served?". All of these issues are fundamental when considering that every such "job" is a strategic agent. It is not only that one avoids uncertainty, knowing the future schedule allows one to make appropriate plans for the interim. Consider a setting where you call up to arrange a 2 hour dental appointment and you are told: "Show up ASAP but there are no guarantees as to when the dentist will see you." This is a non-prompt schedule. It has some disadvantages when compared with the prompt alternative: "Show up at 17:00 and the dentist will see you immediately then."

In this paper we present *prompt* online algorithms that immediately determine as to when an incoming job will be processed (without preemption). The competitive ratio is the best possible, amongst all prompt online algorithms, even if randomization is allowed (the algorithm is in fact deterministic). The competitive ratio compares the sum of completion times of the online algorithm with the [harder to achieve] sum of completion times of an optimal preemptive schedule. Moreover, viewed in the context of jobs being strategic agents, our algorithms are also dominant strategy incentive compatible, and have a particularly simple form. We describe the algorithms in the strategic setting, but – even ignoring strategic issues – no non-trivial online algorithm for the problem of prompt scheduling of jobs was known prior to this study.

Upon arrival, agents are presented with a menu of possible options, where a menu entry is of the form $([b, e], q, \pi)$. This means that the period from b to e is available on machine q and will cost the agent π. These menus are anonymous and do not depend on the agent that arrives. The agent then chooses one of the options. Rational agents will never choose an interval that is shorter than the processing time. (If so the agent cost is ∞).

The cost to the agent is the sum of two components: (a) The time spent waiting, weighted by the agents' [private] weight. *I.e.*, highly impatient agents will have high weight, less impatient agents will have lower weight. (b) The price, π, associated with an option on the menu. Agents seek to minimize their cost.

Consider the case of a single queue, a selfish agent will simply join the queue immediately upon arrival, there is no reason to delay. Thus, jobs will be processed in first-in-first-out (FIFO) order. However, this may be quite bad in terms of the sum of completion times. Imagine a job with processing time L, arriving at time zero, followed by $\sqrt{L}$ jobs of processing time 1, all of which arrive immediately after the first.

As the first job will only be done at time L, the sum of completion times for these $1 + \sqrt{L}$ jobs is about $L^{3/2}$. Contrariwise, if the $\sqrt{L}$ size one jobs were processed before the size L job, the sum of completion times would be about $2L$. Obviously it seems a good idea to delay longer jobs and expedite shorter jobs.

[1] This is illustrated in the introduction of the full version [6].

Similarly, consider a first batch of L jobs, each of size 1 and weight 1, immediately followed by a single job of size 1 and weight W. For FIFO processing, the weighted sum of completion times is $L^2/2$ (for the weight 1 jobs) plus $(L+1) \cdot W$ (for the job of weight W). Optimally, the weight W job should be processed first, followed by the size 1 jobs. The weighted sum of completion times is then about $W + L^2/2$. For any constant L and sufficiently large W, the ratio between the two sums approaches $L+1$.

The main question addressed in this paper is how to produce such dynamic menus so as to incentivize selfish agents towards behavior that achieves some desirable social goal, specifically, minimizing the sum of completion times. The dynamic menu is produced based on the past decisions of the previous agents and the current time[2].

We measure the quality of the solution achieved by the competitive ratio, the ratio between the sum of completion times of the selfish agents, when presented with the dynamic menus, and the minimal sum of completion times, when the future arrivals and their private values are known and there are no incentive considerations. In fact, the comparison is with the optimal preemptive schedule (which could definitely be better than the optimal non-preemptive schedule).

We consider several scenarios:

1. All agents have weight 1 and arbitrary processing times, nothing known apriori on the processing times. This models cases where all agents are equally impatient but have different processing requirements. The underlying idea here is to offer menu options that delay longer jobs so that they do not overly delay many shorter jobs that arrive later.
2. All agents have processing time 1 and arbitrary weight, nothing known apriori on the weights. The underlying idea here is to set prices so as to delay jobs of small weight and thus to allow later jobs of large weight to finish early.
3. Jobs with arbitrary processing times and weights bounded by a known bound $B_{\max}$. This means that we have to delay long jobs and simultaneously have to leave available time slots for jobs with large weights.

The competitive ratios for the different scenarios appear in Table 1. We remark that the lower bounds hold even if one assumes that the machines used are arbitrarily faster than the machines used by the optimal schedule that minimizes the sum of weighted completion times.

1.1 Related Work

For one machine, weighted jobs, available at time zero, ordering the jobs in order of weight/-processing time minimizes the sum of competition times [26]. For one machine, unweighted jobs with release times, a preemptive schedule that always processes the job with the minimal remaining processing time minimizes the sum of weighted completion times [24, 23]. As an offline problem, where jobs cannot be executed prior to some earliest time, finding an optimal non-preemptive schedule is computationally hard [12].

For parallel machines, where jobs arrive over time, a preemptive schedule that always processes the jobs with the highest priority – weight divided by remaining processing time – is a 2 approximation [20]. This algorithm is called *weighted shortest remaining processing time* (WSRPT). If all weights are one this preemptive algorithm is called *shortest remaining processing time* (SRPT). Other online and offline algorithms to minimize the sum of completion times appear in [1, 25, 12].

[2] For clarity we describe the menu as though it was infinite. In fact, one can think of the process as though the menu is presented entry by entry. The selfish job will provably choose an option early on.

Table 1 Competitive Ratios of our Dynamic Menus, and associated lower bounds. $P_{\max}$ is the longest job processing time in the input sequence, it is not known apriori. $W_{\max}$ is the maximal job weight in the sequence, it is not known apriori. $B_{\max}$ is an apriori upper bound on $W_{\max}$.

Processing Time	Job Weight	Menu entries	Upper Bound (Deterministic)	Lower Bound (Randomized)
$p_j \in \mathbb{Z}^+$	$w_j = 1$	intervals (various lengths) no prices	$O(\log P_{\max})$	$\Omega(\log P_{\max})$
$p_j = 1$	$w_j \in \mathbb{Z}^+$	unit length intervals with prices	$O\left(\begin{matrix}\log W_{\max}\cdot\\ (\log\log W_{\max} + \log n)\end{matrix}\right)$	$\Omega(\log W_{\max})$
$p_j \in \mathbb{Z}^+$	$w_j \in \mathbb{Z}^+$	intervals (various lengths) with prices	$O\left(\begin{matrix}\log B_{\max}\cdot\\ (\log P_{\max} + \log n)\end{matrix}\right)$	$\Omega\left(\max\left(\begin{matrix}\log B_{\max},\\ \log P_{\max}\end{matrix}\right)\right)$

[22] show how to convert a preemptive online algorithm into a non-preemptive online algorithm while increasing the completion time of the job by no more than a constant factor. This transformation strongly depends on not determining immediately when the job will be executed. This is in comparison to a prompt algorithm that determines when the job is executed immediately upon the job‘s arrival.

When selfish agents are involved, it is valuable to keep things simple [13]. Offering selfish agents an anonymous menu of options is an example of such a simple process. More complicated mechanisms require trust on the part of the agents.

Recently, [7] considered a similar question to ours, where a job with private processing time had to choose between multiple FIFO queues, where the servers had different speeds. Here, dynamic posted prices were associated with every queue, with the goal of [approximately] minimizing the makespan, the length of time until the last job would finish. Shortly thereafter, [15] used dynamic pricing to minimize the maximal flow time. Dynamic pricing schemes were considered for non-scheduling cost minimization problems in [3].

A constant approximation mechanism for minimizing sum of completion times for selfish jobs was considered in [9], where the setting was an offline setting, the processing time was known in advance and the weight was private information. In an online setting, [14] show a constant approximation preemptive mechanism that gives an $O(1/\epsilon^2)$ approximation to the sum of flow times when using machines that are faster by a factor of $1 + \epsilon$.

Online mechanisms were considered in [17, 8], whereas prompt online mechanisms are defined in [4].

In this paper our goals are pricing schemes that affect agents as to behave in a manner that [approximately] minimizes the sum of weighted completion times.

There is a vast body of work on machine scheduling problems, in offline and online settings, with strategic agents involved and not, and in a host of models. It is impossible to do justice to this body of work but a very short list of additional relevant papers includes [10, 19, 11, 18, 21, 2, 16].

1.2 Organization of this paper

In Section 2 we describe our model. In Section 3 we give an optimal $O(\log P_{\max})$-competitive menu based mechanism for the case of arbitrary [unknown] lengths and identical weights, and in Section 4 we show a matching $\Omega(\log P_{\max})$ lower bound that holds for any [randomized, non-truthful] prompt online algorithm. In the full version [6] we handle the case of arbitrary weighted jobs with identical processing times and give a $O(\log W_{\max}(\log\log W_{\max} + \log n))$-competitive pricing menu. An $\Omega(W_{\max})$ lower bound for this case is given.

2 The Model

We consider a job scheduling setting with m machines and n jobs that arrive in real time, where p_j, w_j, and r_j are, respectively, the processing time, weight, and release time of the jth job to arrive. It may be that $r_j = r_{j+1}$, *i.e.*, more than one job arrive at the same time. However, job decisions are made sequentially in index order.

A valid input for this problem can be described as a sequence of jobs

$$\sigma = (r_1, w_1, p_1), (r_2, w_2, p_2), \ldots, (r_n, w_n, p_n),$$

where the *release time* $r_i \le r_{i+1}$ for $i = 1, \ldots, n-1$, the *job weight* $w_i \ge 1$ for $i = 1, \ldots, n$, and the job *processing time* $p_i \ge 1$ for $i = 1, \ldots, n$. We refer to the jth job in this sequence as job j. We use the terms *size* and processing time interchangeably. Let $\sigma[1..\ell]$ be the length ℓ prefix of σ. The total volume of a set of jobs D, denoted $vol(D)$ is the sum of processing times of the jobs in D, i.e., $vol(D) = \sum_{j \in D} p_j$.

Let $s_j \ge r_j$ be the time at which job j starts processing (on some machine $1 \le q \le m$). The completion time of job j is $c_j = s_j + p_j$.

The objective considered in this paper is to minimize the sum of [weighted] completion times; *i.e.*, we wish to minimize $\sum_{j=1}^{n} w_j \cdot c_j$.

For jobs j, j', with $j < j'$ and with $r_j = r_{j'}$, job j is assigned (or chooses) machine q_j at time s_j before job j' is assigned machine $q_{j'}$ at $s_{j'}$. We say that (q_j, s_j) and $(q_{j'}, s_{j'})$ *overlap*, if $q_j = q_{j'}$ and ($s_j \le s_{j'} < c_j = s_j + p_j$ or $s_{j'} \le s_j < c_{j'} = s_{j'} + p_{j'}$).

A *valid* (non-preemptive) schedule for an input σ is a sequence

$$(m_1, s_1), (m_2, s_2), \ldots, (m_n, s_n)$$

where no overlaps occur. An *online* algorithm determines (m_j, s_j) after seeing $\sigma[1 \ldots j]$ and before seeing job $j+1$.

We consider online mechanisms where jobs are selfish agents, processing times and weights are private information, and job j is presented with a menu of options upon arrival. Every option on the menu is of the form (I, q, π) where (i) I is a time interval $[b(I), e(I)]$, with integer endpoints, and where $b(I) \ge r_j$, (ii) $1 \le q \le m$ is some machine, and (iii) π is the price for choosing this entry. The menu of options presented to job j is computed after jobs $1, \ldots, j-1$ have all made their choices and also depends on the release time of job j, r_j (because one cannot process a job in the past). We assume *no feedback* from jobs after they choose their menu options, i.e., if a job of size p chooses an interval I of length $|I| > p$ on some machine, we do not know the interval is only partly used, and specifically, cannot offer the $|I| - p$ remaining to future jobs.

For job j that chooses menu entry $([b(I), e(I)], q, \pi)$ we use the following notation (i) $I(j)$ for the interval chosen by job j, $[b(I), e(I)]$, (ii) $M(j)$ for the machine chosen by job j, q, and (iii) $\Pi(j)$ for the price of the entry chosen by j, π.

Although the menus we describe are infinite, one can present the menu items sequentially. With unit weight jobs, a job of processing time p will make its choice within the first $\log p$ options presented. With unit length jobs, a job of weight w will make its choice within the first $\log w$ options presented. With arbitrary lengths and arbitrary weights, a job of processing time p and of weight w will make its choice within the first $\log p \cdot \log w$ options presented.

The cost to job j with weight w_j and processing time p_j for choosing the menu entry $([b, e], q, \pi)$ is ∞ if the time interval is too short: $e - b < p_j$. If $e - b \ge p_j$ then the cost to job j is a cost of w_j for every unit of time until job j starts processing, plus the extra price from the menu. *I.e.*, the cost to job j with release time r_j, processing time p_j and weight w_j, for choosing menu entry $([b, e], q, \pi)$, $e - b \geq p_j$, is

$$(b + p_j) \cdot w_j + \pi.$$

For the specialized cases of weight one jobs or unit length jobs the general model above is somewhat simpler:

2.1 Modeling weight one jobs with arbitrary Processing times

If jobs have weight one, we give (optimal) menus that do not require pricing menu entries. Any entry on the menu is available for free. Therefore, we can simplify the menu structure as follows: The job chooses a time interval and a machine from a menu with entries of the form $([b, e], 1 \leq q \leq m)$ where the first entry is a time interval, and the second entry is a machine[3].

Jobs choose from the menu one of the entries immediately upon arrival. As above, we say that job j chooses menu entry $(I(j), M(j))$ where $I(j)$ is an interval, and $1 \leq M(j) \leq m$.

For job j with arrival time r_j, and processing time p_j the cost associated with choosing the menu item $([b, e], 1 \leq q \leq m)$ is ∞ if $p_j > e - b$ and $(b + p_j)$ otherwise. Jobs always seek to minimize their cost.

2.2 Modeling unit length jobs of arbitrary weight

Every job requires one unit of processing time on one of m different processors. Every job j is a selfish agent that has a private weight w_j, the cost to the job of one unit of delay.

The job chooses a machine and time slot from a menu with entries of the form $([i, i+1], 1 \leq q \leq m, \pi)$ where the first entry is a time slot, the second entry is a machine, and the third entry is the price of this time slot on the machine.

Jobs choose from the menu one of the entries immediately upon arrival. Job j is said to choose menu item $(I(j), M(j), \Pi(j))$ where $I(j)$ is a length one interval, $1 \leq M(j) \leq m$, and $\Pi(j)$ is the price to be paid for choosing this option.

For job j with arrival time r_j, and weight w_j the cost associated with choosing the menu item $([i, i + 1], 1 \leq q \leq m, \pi)$ is $w_j(i + 1) + \pi$. Jobs always seek to minimize their cost.

3 Dynamic Menu for Jobs with Heterogeneous Processing Times

In this section we introduce a dynamic menu based mechanism, for jobs of weight one and heterogeneous processing times, with competitive ratio $O(\log P_{\max})$, where $P_{\max}$ is the maximal job processing time among all jobs. Due to lack of space, the analysis of the mechanism is deferred to the full version [6].

In Section 3.1 we provide integer sequences and corresponding interval sequences that serve as a building block for our dynamic menu mechanism, which is presented in Section 3.2.

3.1 The S_k Integer and Interval Sequences

We define sequences of integers S_k, $k = 0, 1, \ldots$, as follows: Let $S_0 = \langle 1 \rangle$ and for $k > 0$ let $S_k = S_{k-1} \| S_{k-1} \| \langle 2^k \rangle$ where $\|$ denotes concatenation. Ergo,

$$S_0 = \langle 1 \rangle; \quad S_1 = S_0 \| S_0 \| \langle 2^1 \rangle = \langle 1, 1, 2 \rangle; \quad S_2 = S_1 \| S_1 \| \langle 2^2 \rangle = \langle 1, 1, 2, 1, 1, 2, 4 \rangle; \quad \cdots$$

[3] Although the general setting allows pricing menu items, it turns out that for weight 1 jobs the optimal menu does not need to differentiate entries by price.

Let $n_k = 2^{k+1} - 1$ denote the length of S_k (follows inductively from $n_0 = 1$ and $n_k = 2n_{k-1} + 1$). Let $S_k[i]$, $i = 1, \ldots, n_k$ be the ith element of S_k. Let S_∞ be an infinite sequence whose length n_k prefix is S_k (for all k):

$$S_\infty = \langle 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 16, 1, \ldots \rangle.$$

Let $S_\infty[i]$, $i = 1, 2, \ldots$ be the ith element of S_∞. Note that $S_k[i] = S_{k'}[i]$ for all $k \le k'$ and all $i = 1, \ldots, n_k$, ergo, S_k is a prefix of $S_{k'}$ for $k \le k'$.

▶ **Lemma 1.** *For all $d \ge 0$, for all $0 \le k \le d$, the sum of all the 2^k value items in S_d is equal 2^d. That is,* $\sum_{1 \le i \le n_d : S_d[i] = 2^k} 2^k = 2^d$.

We use the S_k sequences to define interval sequences. Let γ_i be the sum of the first i entries in S_∞, $\gamma_i = \sum_{j=1}^{i} S_\infty[j]$ (*i.e.*, $\gamma_1 = 1$, $\gamma_2 = 2$, $\gamma_3 = 4$, etc.).

We define $S_k(t)$, $t \ge 0$, to be a sequence of n_k consecutive intervals, the first of which starts at time t, and where the length of the jth interval equals $S_k[j]$. *I.e.*,

$$S_k(t) = \langle [t, t + \gamma_1], [t + \gamma_1, t + \gamma_2], \ldots, [t + \gamma_{n_k - 1}, t + \gamma_{n_k}] \rangle \, .$$

For example

$$S_2(2) = \langle [2, 3], [3, 4], [4, 6], [6, 7], [7, 8], [8, 10], [10, 14] \rangle. \tag{1}$$

For any interval sequence S let $b(S)$ be the start of the first interval in S and let $e(S)$ be the end of the last interval in S. For example, $b(S_2(2)) = 2$ and $e(S_2(2)) = 14$.

We say that S_k *appears* in $S_d(t)$ if there exists some t' such that the interval sequence $S_k(t')$ is a contiguous subsequence of $S_d(t)$. In this case we also say that $S_k(t')$ appears in $S_d(t)$. Note that while S_k is a sequence of integers, both $S_k(t')$ and $S_d(t)$ are interval sequences.

By construction, for any k and any $t \ne t'$ if $S_k(t)$ and $S_k(t')$ appear in some $S_d(\tilde{t})$, then $[b(S_k(t)), e(S_k(t))]$ and $[b(S_k(t')), e(S_k(t'))]$ are disjoint except for, possibly, their endpoints. Let I be an interval of length 2^k that appears in $S_\infty(t)$. Then there is a unique t' such that $S_k(t')$ appears in $S_\infty(t)$ and I is the last interval of $S_k(t')$. It follows from Lemma 1 that

▶ **Corollary 2.** *For all $k \le d$, for all t,*
1. *S_k appears in $S_d(t)$ 2^{d-k} times.*
2. *The sum of the lengths of the intervals in $S_d(t)$ is $(d + 1)2^d$.*

The interval sequences defined above suggests a new possible static algorithm. Divide the timeline of each machine into intervals as in $S_\infty(0)$, and let any job that arrives occupy the first unoccupied interval it fits in. Unfortunately, when the competitive ratio is evaluated as a function of $P_{\max}$ alone, this algorithm is $\Omega\left(\sqrt{P_{\max}}\right)$ competitive[4] (When the competitive ratio may be a function of $P_{\max}$ and n, this algorithm is $O(\log P_{\max} + \log n)$ competitive as analyzed in Theorem 5 of the full version).

▶ **Definition 3.** A *state* is a vector of consecutive interval sequences of the form

$$\begin{aligned} A &= \langle A_1, A_2, \cdots, A_\ell \rangle \text{ where} \\ A_i &= S_{k_i}(t_i) \text{ for every } 1 \le i \le \ell, \end{aligned}$$

for some ℓ (which we refer to as the *length* of A) and integers k_i for $1 \le i \le \ell$, and where $e(A_i) = e(S_{k_i}(t_i)) \le t_{i+1} = b(A_{i+1})$ for $1 \le i \le \ell - 1$. This means that the interval sequences are disjoint and ordered by their starting times. Note that there might be gaps between two consecutive state entries, i.e., $e(A_i) < b(A_{i+1})$ for some $1 \le i \le \ell - 1$.

[4] This is shown in Appendix B in the full version [6].

3.2 $O(\log P_{\max})$ Competitive Dynamic Menu

When job $j+1$ arrives the algorithm is in some *configuration* $\psi^j = \left(A^j, X^j\right)$, where A^j is some state of length ℓ_j, and X^j is the set of intervals occupied by the previous j jobs. State A^j represents every machines' division of $\left[0, \max_{i\in[j]} c_i\right]$ into time intervals (same division for all machines). This division will be kept at any future time. For every $i < \ell_j$, A^j_i is fixed and will be a part of every future state, while $A^j_{\ell_j}$ might be subject to change. We refer to $A^j_{\ell_j}$ as the *tentative sequence* of state A^j. X^j keeps track of all previously allocated intervals (in all machines): $([b,e],q) \in X^j$ means that some job $j' \le j$ chose the interval $[b,e]$ on machine $1 \le q \le m$. Note that the size of job j', $p_{j'}$, might be strictly smaller than the length of the interval $(e-b)$, yet it is still considered occupied.

We note that our mechanism has the property that, roughly, every time interval in state A_j has a $\frac{1}{\log P_{\max}}$ fraction of its volume allocated to "small" jobs.

Generating the Dynamic Menu

Given a state $A = (A_1, A_2, \ldots, A_\ell)$ and a time t, we define an interval sequence τ as follows:

$$\tau(A,t) = \begin{cases} A_1 \| A_2 \| \ldots \| A_\ell \| S_\infty(t) & t \ge e\left(A_\ell\right) \\ A_1 \| A_2 \| \ldots \| A_{\ell-1} \| S_\infty(b\left(A_\ell\right)) & t < e\left(A_\ell\right) \end{cases}$$

τ is used to create the menu presented to a job j. We present an algorithm for the creation of the menu, based on the previous configuration ψ^{j-1}, and the current time t.

- Let $\tau^j = \tau\left(A^{j-1}, r_j\right)$.
- Set d_1 to be the length of the first time interval in τ^j beginning at time $b_1 \ge t$.
- Add $([b_1, b_1+d_1], q)$ to the menu for all machines $1 \le q \le m$ in which $[b_1, b_1+d_1]$ is unoccupied (i.e, $([b_1, b_1+d_1], q) \notin X^{j-1}$).
- Set $i = 1$
- Repeat until job j chooses an interval:
  - Let d_{i+1} be the length of the first interval longer than d_i in τ^j that starts at time $b_{i+1} \ge t$ (it follows that $b_{i+1} > b_i$).
  - Add $([b_{i+1}, b_{i+1}+d_{i+1}], q)$ to the menu for all machines $1 \le q \le m$ in which $[b_{i+1}, b_{i+1}+d_{i+1}]$ is unoccupied (i.e., $([b_{i+1}, b_{i+1}+d_{i+1}], q) \notin X^{j-1}$).
  - Set $i = i+1$.

By construction, no job will ever choose a time interval that starts before the job arrival time, nor will it ever choose a slot that has already been chosen.

A selfish job of length p_j always chooses a menu entry of the form $([b,e],q)$ where b is the earliest menu entry with $p_j \le e-b$.

Updating States

After job j makes its choice of menu entry, $(I(j), M(j))$, we update the configuration from $\psi^{j-1} = \left(A^{j-1}, X^{j-1}\right)$ to $\psi^j = \left(A^j, X^j\right)$. Clearly, $X^j = X^{j-1} \cup \{(I(j), M(j))\}$. In the rest of this section we describe how to compute A^j.

Recall that a state is a vector of consecutive and disjoint interval sequences. Initially, $A^0 = \langle\rangle$ with length $\ell_0 = 0$ and $A^0_{\ell_0}$ is an empty sequence with $b\left(A^0_{\ell_0}\right) = e\left(A^0_{\ell_0}\right) = 0$. A^j always contains all of A^{j-1}'s interval sequences except possibly the tentative sequence $A^{j-1}_{\ell_{j-1}}$. When job j of size 2^k chooses an interval, the new tentative sequence $A^j_{\ell_j}$ can be one of the following:

Table 2 Update rules: After job j makes its choice (and c_j is determined), the new state A^j is a function of (i) A^{j-1}, (ii) release time r_j, (iii) processing time $p_j = 2^k$, and (iv) completion time c_j.

	ℓ_j	$A^j_{\ell_j}$	$c_j \le e\left(A^{j-1}_{\ell_{j-1}}\right)$	$r_j \ge e\left(A^{j-1}_{\ell_{j-1}}\right)$	$A^{j-1}_{\ell_{j-1}} = S_d(t)$ $k \le d$
1	ℓ_{j-1}	$A^{j-1}_{\ell_{j-1}}$	True	-	-
2	$\ell_{j-1}+1$	$S_k(r_j)$	False	True	-
3	$\ell_{j-1}+1$	$S_k\left(e\left(A^{j-1}_{\ell_{j-1}}\right)\right)$	False	False	True
4	ℓ_{j-1}	$S_k\left(b\left(A^{j-1}_{\ell_{j-1}}\right)\right)$	False	False	False

1. *Unchanged from former:* The new tentative sequence in A^j is the same as the former tentative sequence in A^{j-1}, *i.e.*, $A^j_{\ell_j} = A^{j-1}_{\ell_{j-1}}$. This happens when $I(j) \in A^{j-1}$, see entry 1 in Table 2.
2. *Disjoint from former:* The former tentative sequence, $A^{j-1}_{\ell_{j-1}}$ becomes *fixed*, and the new tentative sequence $A^j_{\ell_j}$ is disjoint from the former. The tentative sequence in A^{j-1}, $A^{j-1}_{\ell_{j-1}}$, is the ℓ_{j-1}th element in all future states A^i, for $i \ge j$. See entries 2 and 3 in Table 2.
3. *Extension of former:* The new tentative sequence is an *extension* of the former tentative sequence. *I.e.*, if $A^{j-1}_{\ell_{j-1}} = S_d(t)$ then $\ell_j = \ell_{j-1}$ and $A^j_{\ell_j} = S_k(t), k > d$. See entry 4 in Table 2.

Let A^j_i, A^j_{i+1} be two consecutive interval sequences in a state A^j. If $b\left(A^j_{i+1}\right) > e\left(A^j_i\right)$, we say the interval $\left[e\left(A^j_i\right), b\left(A^j_{i+1}\right)\right]$ is a *gap*.

Figure 1 is an example with 5 jobs that arrive over time, and the matching configuration changes. The jobs in Figure 1 illustrate cases 1–4 from Table 2 in the following order: case 2 for job 1, case 1 for job 2, case 3 for job 3, case 4 for job 4 and case 2 for job 5.

3.2.1 High level overview of the analysis

Due to lack of space, we only give a high level overview of the analysis. The full analysis appears in Section 3.4 of the full version [6]. We first show that w.l.o.g. one may assume that all job lengths are powers of 2 and that the adversary's schedule never includes gaps. In our analysis, we compare the completion time of each job under our mechanism with the completion time of the same job under SRPT. Let j be a job in the input sequence. We define $D(j) = \{j' \le j | p_{j'} \le p_j\}$ to be the set of all jobs that arrived no later than job j and that are no bigger (note $j \in D(j)$). These jobs are all completed no later than job j both under our mechanism and under SRPT, implying $c^*_j \ge \frac{1}{m} vol(D(j))$ (where c^*_j is the completion time of job j under SRPT). Our analysis is based upon this observation. We show that the mechanism depicted above ensures that $c_j = O(\log P_{\max}) \cdot c^*_j$ for every job j. Since SRPT is an $O(1)$-competitive algorithm, this immediately implies the following.

▶ **Theorem 4.** *Our mechanism is $O(\log P_{\max})$-competitive.*

3.3 Arbitrary processing times, weight $\le B_{\max}$

The static algorithm suggested at the end of Section 3.1 used for weight one jobs of arbitrary sizes can be easily adapted to weights in some predetermined range from 1 to $B_{\max}$. Replicate every interval in the sequence $S_\infty(0)$ $\log B_{\max} + 1$ times. For $\ell = 0, \dots, \log B_{\max}$, the ℓth copy is designed to hold only jobs of weight $\ge 2^\ell$. To achieve this, one associates prices with

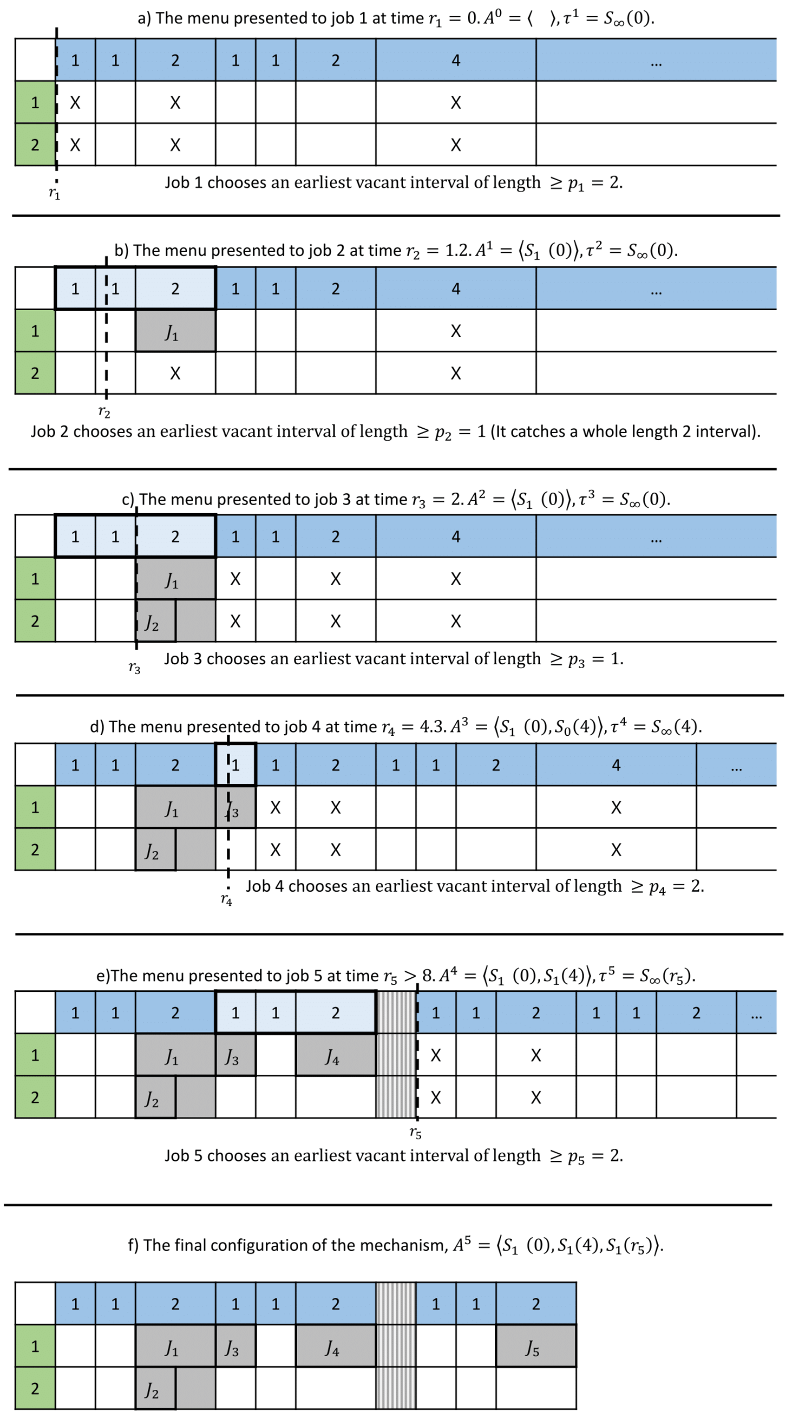

a) The menu presented to job 1 at time $r_1 = 0. A^0 = \langle\ \rangle, \tau^1 = S_\infty(0)$.
Job 1 chooses an earliest vacant interval of length $\geq p_1 = 2$.
b) The menu presented to job 2 at time $r_2 = 1.2. A^1 = \langle S_1\ (0)\rangle, \tau^2 = S_\infty(0)$.
Job 2 chooses an earliest vacant interval of length $\geq p_2 = 1$ (It catches a whole length 2 interval).
c) The menu presented to job 3 at time $r_3 = 2. A^2 = \langle S_1\ (0)\rangle, \tau^3 = S_\infty(0)$.
Job 3 chooses an earliest vacant interval of length $\geq p_3 = 1$.
d) The menu presented to job 4 at time $r_4 = 4.3. A^3 = \langle S_1\ (0), S_0(4)\rangle, \tau^4 = S_\infty(4)$.
Job 4 chooses an earliest vacant interval of length $\geq p_4 = 2$.
e)The menu presented to job 5 at time $r_5 > 8. A^4 = \langle S_1\ (0), S_1(4)\rangle, \tau^5 = S_\infty(r_5)$.
Job 5 chooses an earliest vacant interval of length $\geq p_5 = 2$.
f) The final configuration of the mechanism, $A^5 = \langle S_1\ (0), S_1(4), S_1(r_5)\rangle$.
r_1
r_2
r_3
r_4
r_5
J_1
J_2
J_3
J_4
J_5

Figure 1 Changing Menus of the Dynamic Menu Algorithm, as jobs arrive and make choices. The two bottom rows in the tables represents two machines. An X in a machine cell represents an (interval,machine) entry in the currently presented menu. A dashed line marks the release time of the current job. Gray cells represent choices previously made by jobs. A gap is represented by a rectangle filled with vertical lines. A rectangle outline in the top row of a table represents the tentative sequence before job j makes it choice, i.e., $A^{j-1}_{\ell_{j-1}}$. Note that this example does not make the simplicity assumptions made in the analysis.

such intervals, as done in Section 5 of the full version. The analysis preformed in Appendix A of the full version holds when multiplying every element with $\log B_{\max} + 1$, implying a competitive ratio of $O\left((\log P_{\max} + \log n) \cdot \log B_{\max}\right)$.

4 Lower Bound on the Competitive Ratio for any Prompt Online Algorithm, Arbitrary Lengths

We now show that any prompt online scheduling algorithm must have a competitive ratio of $\Omega\left(\log P_{\max}\right)$, even if randomization is allowed.

Let c be the competitive ratio of some algorithm ALG as a function of $P_{\max}$. Consider the following sequence, for $\boldsymbol{P}$ to be determined later:

For $i = 0, \ldots, 16c$:
- $n_i = 2^i$ jobs of size $P_i = \frac{\boldsymbol{P}}{2^i}$ arrive one after the other (at time 0).
- If the expected number of P_i sized jobs with completion time greater than $8c\boldsymbol{P}$ is at least $n_i/2$, stop the sequence. Let j be the last iteration.

Note that for every $i = 0, \ldots, 16c$ it holds that $n_i \cdot P_i = \boldsymbol{P}$.

▶ **Lemma 5.** *There must be an iteration $j \in \{0, \ldots, 16c\}$ for which in expectation more than half of the jobs have completion time greater than $8c\boldsymbol{P}$.*

Proof. Let X_i be a random variable representing the number of size P_i jobs, with completion time greater than $8c\boldsymbol{P}$. If for all $i \in \{0, \ldots, 16c\}$, $\mathbb{E}\left[X_i\right] \leq n_i/2$, then the total expected volume of jobs completed before time $8c\boldsymbol{P}$ is at least

$$\sum_{i=0}^{16c} \mathbb{E}\left[(n_i - X_i) \cdot P_i\right] \geq \sum_{i=0}^{16c} \frac{n_i}{2} \cdot P_i = \sum_{i=0}^{16c} \frac{\boldsymbol{P}}{2} > 8c\boldsymbol{P},$$

a contradiction. ◀

▶ **Theorem 6.** *Any random prompt online algorithm must be $\Omega\left(\log P_{\max}\right)$ competitive for the above sequence.*

Proof. According to Lemma 5, there must be some $j \in \{0, \ldots, 16c\}$ for which in expectation at least half of the jobs are completed after time $8c\boldsymbol{P}$. Given this j, we give bounds on both OPT and ALG. Let X_i be as in Lemma 5. In ALG, $\mathbb{E}\left[X_j\right] > n_j/2$, thus:

$$\mathbb{E}\left[\mathrm{Cost}(\mathrm{ALG})\right] > \mathbb{E}\left[X_j \cdot 8c\boldsymbol{P}\right] > 8c\boldsymbol{P} \cdot \frac{n_j}{2} = 4c\boldsymbol{P} \cdot n_j. \tag{2}$$

In OPT, the jobs are scheduled from the smallest one (of size P_j) to the biggest one (of size $P_0 = \boldsymbol{P}$). The kth job of size P_i to be scheduled, is completed after all jobs smaller than it (of sizes $P_{i+1}, \ldots, P_j$) and after $k-1$ jobs of size P_i, and therefore has a completion time of

$$\left(\sum_{\ell=i+1}^{j} n_\ell \cdot P_\ell\right) + P_i \cdot (k-1) + P_i = P_i \cdot k + \sum_{\ell=i+1}^{j} n_\ell \cdot P_\ell.$$

Summing over all jobs of all sizes, we have

$$\begin{aligned}\mathrm{Cost}(\mathrm{OPT}) &= \sum_{i=0}^{j}\left(\sum_{k=1}^{n_i}\left(P_i\cdot k+\sum_{\ell=i+1}^{j} n_\ell\cdot P_\ell\right)\right)\\ &= \underbrace{\sum_{k=1}^{n_j}P_j\cdot k}_{(i)}+\underbrace{\sum_{i=0}^{j-1}\sum_{k=1}^{n_i}P_i\cdot k}_{(ii)}+\underbrace{\sum_{i=0}^{j-1}\left(\sum_{\ell=i+1}^{j}n_\ell\cdot P_\ell\right)\cdot n_i}_{(iii)}. \qquad (3)\end{aligned}$$

We now bound each term of Cost(OPT) separately.

$$(i): P_j\sum_{k=1}^{n_j}k < P_j\cdot n_j^2=\boldsymbol{P}\cdot n_j. \qquad (4)$$

$$(ii): \sum_{i=0}^{j-1}P_i\sum_{k=1}^{n_i}k<\sum_{i=0}^{j-1}P_i\cdot n_i^2=\boldsymbol{P}\cdot\sum_{i=0}^{j-1}2^i\le\boldsymbol{P}\cdot 2^j=\boldsymbol{P}\cdot n_j. \qquad (5)$$

For (iii) we have

$$\begin{aligned}(iii): \sum_{i=0}^{j-1}\left(\sum_{\ell=i+1}^{j}n_\ell\cdot P_\ell\right)\cdot n_i &= \sum_{i=0}^{j-1}\sum_{\ell=i+1}^{j}\boldsymbol{P}\cdot 2^i=\boldsymbol{P}\sum_{i=0}^{j-1}(j-i)2^i=\boldsymbol{P}\sum_{i=1}^{j}i\cdot 2^{j-i}\\ &= \boldsymbol{P}\cdot 2^j\sum_{i=1}^{j}\frac{i}{2^i}\le 2\boldsymbol{P}\cdot n_j. \qquad (6)\end{aligned}$$

From Equations (4), (5) and (6), we get that $\mathrm{Cost}(\mathrm{OPT})\le 4\boldsymbol{P}\cdot n_j$. Therefore,

$$\mathbb{E}\left[\mathrm{Cost}(\mathrm{ALG})/\mathrm{Cost}(\mathrm{OPT})\right]>c,$$

in contradiction to the assumption that ALG is c-competitive.

For the input sequence to be valid, it must be that $P_j\ge 1$. As $j\le 16c$, it is sufficient that $c(\boldsymbol{P})\le\frac{1}{16}\log\boldsymbol{P}$, as in this case, $P_{16c}=\frac{\boldsymbol{P}}{2^{16c}}\ge 1$. So for every competitive ratio function c such that $c(P_{\max})=o(\log P_{\max})$ there exists a sufficiently large $\boldsymbol{P}$ for which $c(\boldsymbol{P})\le\frac{1}{16}\log\boldsymbol{P}$, and our input is a valid counter example. ◀

References

1 J. Bruno, E. G. Coffman, Jr., and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Commun. ACM*, 17(7):382–387, 1974. doi:10.1145/361011.361064.

2 George Christodoulou, Elias Koutsoupias, and Akash Nanavati. Coordination mechanisms. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, pages 345–357, 2004. doi:10.1007/978-3-540-27836-8_31.

3 Ilan Reuven Cohen, Alon Eden, Amos Fiat, and Lukasz Jez. Pricing online decisions: Beyond auctions. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 73–91. SIAM, 2015. doi:10.1137/1.9781611973730.7.

4 Richard Cole, Shahar Dobzinski, and Lisa Fleischer. Prompt mechanisms for online auctions. In *Proceedings of the 1st International Symposium on Algorithmic Game Theory*, SAGT '08, pages 170–181, Berlin, Heidelberg, 2008. Springer-Verlag. doi:10.1007/978-3-540-79309-0_16.

5 Constantinos Daskalakis, Moshe Babaioff, and Hervé Moulin, editors. *Proceedings of the 2017 ACM Conference on Economics and Computation, EC '17, Cambridge, MA, USA, June 26-30, 2017*. ACM, 2017.

6 Alon Eden, Michal Feldman, Amos Fiat, and Tzahi Taub. Prompt scheduling for selfish agents. *CoRR*, abs/1804.03244, 2018. `arXiv:1804.03244`.

7 Michal Feldman, Amos Fiat, and Alan Roytman. Makespan minimization via posted prices. In Daskalakis et al. [5], pages 405–422. `doi:10.1145/3033274.3085129`.

8 Eric J. Friedman and David C. Parkes. Pricing wifi at starbucks: Issues in online mechanism design. In *Proceedings of the 4th ACM Conference on Electronic Commerce*, EC '03, pages 240–241, New York, NY, USA, 2003. ACM. `doi:10.1145/779928.779978`.

9 Vasilis Gkatzelis, Evangelos Markakis, and Tim Roughgarden. Deferred-acceptance auctions for multiple levels of service. In Daskalakis et al. [5], pages 21–38. `doi:10.1145/3033274.3085142`.

10 R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

11 Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

12 Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Math. Oper. Res.*, 22(3):513–544, 1997. `doi:10.1287/moor.22.3.513`.

13 Jason D. Hartline and Tim Roughgarden. Simple versus optimal mechanisms. In *Proceedings 10th ACM Conference on Electronic Commerce (EC-2009), Stanford, California, USA, July 6–10, 2009*, pages 225–234, 2009. `doi:10.1145/1566374.1566407`.

14 Sungjin Im and Janardhan Kulkarni. Fair online scheduling for selfish jobs on heterogeneous machines. In Christian Scheideler and Seth Gilbert, editors, *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 185–194. ACM, 2016. `doi:10.1145/2935764.2935773`.

15 Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Clifford Stein. Minimizing maximum flow time on related machines via dynamic posted pricing. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 51:1–51:10. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. `doi:10.4230/LIPIcs.ESA.2017.51`.

16 Nicole Immorlica, Li (Erran) Li, Vahab S. Mirrokni, and Andreas S. Schulz. Coordination mechanisms for selfish scheduling. *Theor. Comput. Sci.*, 410(17):1589–1598, 2009. `doi:10.1016/j.tcs.2008.12.032`.

17 Ron Lavi and Noam Nisan. Competitive analysis of incentive compatible on-line auctions. In *Proceedings of the 2Nd ACM Conference on Electronic Commerce*, EC '00, pages 233–241, New York, NY, USA, 2000. ACM. `doi:10.1145/352871.352897`.

18 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990. `doi:10.1007/BF01585745`.

19 J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. In P.L. Hammer, E.L. Johnson, B.H. Korte, and G.L. Nemhauser, editors, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 343–362. Elsevier, 1977. `doi:10.1016/S0167-5060(08)70743-X`.

20 Nicole Megow and Andreas S. Schulz. On-line scheduling to minimize average completion time revisited. *Oper. Res. Lett.*, 32(5):485–490, 2004. `doi:10.1016/j.orl.2003.11.008`.

21 Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35(1-2):166–196, 2001.

22 Cynthia Phillips, Clifford Stein, and Joel Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82(1):199–223, Jun 1998. `doi:10.1007/BF01585872`.

23 Linus Schrage. Letter to the editor-a proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

24 Linus E. Schrage and Louis W. Miller. The queue m / g /1 with the shortest remaining processing time discipline. *Operations Research*, 14(4):670–684, 1966.

25 David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. *SIAM J. Comput.*, 24(6):1313–1331, 1995. `doi:10.1137/S0097539793248317`.

26 Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956. `doi:10.1002/nav.3800030106`.

Weighted Model Counting on the GPU by Exploiting Small Treewidth

Johannes K. Fichte
International Center for Computational Logic, TU Dresden, 01062 Dresden, Germany
johannes.fichte@tu-dresden.de
https://orcid.org/0000-0002-8681-7470

Markus Hecher
Institute of Logic and Computation, TU Wien, Favoritenstraße 9-11, 1040 Wien, Austria
hecher@dbai.tuwien.ac.at
https://orcid.org/0000-0003-0131-6771

Stefan Woltran
Institute of Logic and Computation, TU Wien, Favoritenstraße 9-11, 1040 Wien, Austria
woltran@dbai.tuwien.ac.at
https://orcid.org/0000-0003-1594-8972

Markus Zisser
Institute of Logic and Computation, TU Wien, Favoritenstraße 9-11, 1040 Wien, Austria
markus.zisser@student.tuwien.ac.at

Abstract

We propose a novel solver that efficiently finds almost the exact number of solutions of a Boolean formula (#SAT) and the weighted model count of a weighted Boolean formula (WMC) if the treewidth of the given formula is sufficiently small. The basis of our approach are dynamic programming algorithms on tree decompositions, which we engineered towards efficient parallel execution on the GPU. We provide thorough experiments and compare the runtime of our system with state-of-the-art #SAT and WMC solvers. Our results are encouraging in the sense that also complex reasoning problems can be tackled by parameterized algorithms executed on the GPU if instances have treewidth at most 30, which is the case for more than half of counting and weighted counting benchmark instances.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms, Theory of computation → Complexity theory and logic, Computer systems organization → Single instruction, multiple data, Hardware → Theorem proving and SAT solving, Computing methodologies → Graphics processors

Keywords and phrases Parameterized Algorithms, Weighted Model Counting, General Purpose Computing on Graphics Processing Units, Dynamic Programming, Tree Decompositions, Treewidth

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.28

Funding The work has been supported by the Austrian Science Fund (FWF), Grants Y698 and P26696, and the German Science Fund (DFG), Grant HO 1294/11-1. The first and second author are also affiliated with the University of Potsdam, Germany.

© Johannes K. Fichte, Markus Hecher, Stefan Woltran, and Markus Zisser;
licensed under Creative Commons License CC-BY

1 Introduction

Many computational problems in modern society account to probabilistic reasoning, statistics, and combinatorics. Examples of such problems are identifying the reliability of energy infrastructure [16] or learning preference distributions [10]. Several of these real-world problems can be solved by *representing* the question in (Boolean) formulas [42, 15, 47] and associating the number of solutions of the formula directly with the answer to the question. The task to compute the number of solutions of a formula is usually referred to as the problem #SAT, which is theoretically of high worst case complexity (#P-hard [38]), and generalizes the problem of deciding whether a formula has a solution (SAT). If in addition each literal in the formula has an associated weight and we are interested in the sum of weights of all solutions, where the weight of a truth assignment is the product of the weights of its literals, we speak about weighted model counting (WMC).

One approach to tackle these problems origins in parameterized algorithms, which are based on the assumption that certain structural restrictions in the input allow for efficient solving of problems that are hard in general. A seminal example in this direction is to exploit small treewidth for SAT and #SAT [40]. *Treewidth* roughly measures the tree-likeness of an input graph and is defined in terms of certain decompositions of the graph. For Boolean formulas one takes a graph representation of the input formula, namely the *primal* or *incidence* graph. In order to solve #SAT, dynamic programming on a tree decomposition of the graph representation [40] is used. There one traverses the decomposition in post-order (bottom-up traversal) and computes at each node information stored in a *table*. The runtime heavily depends on the size of the table, which is bounded by a function in the treewidth. Recent competitions in parameterized complexity [14] reveal that exact parameterized algorithms are not just a vibrant theoretical research area, but their implementations are also able to outperform up-to-date SAT solvers when determining treewidth.

State-of-the-art #SAT or WMC engines so far rely on standard techniques from SAT-solving [44, 41, 26], knowledge compilation [33], or approximate solving [7, 8] by means of sampling using SAT solvers. There is few work on parallelizing certain aspects of modern SAT solving on Graphics Progressing Units (GPUs), e.g., [11]. However, a core technique of SAT solving, conflict driven clause learning (CDCL), has inherent sequential aspects and does not parallelize well [3, 22, 24, 34]. In contrast, many problems in artificial intelligence and machine learning have significantly benefited from parallelization. In particular, running algorithms on GPUs or using special purpose processing units such as Tensor Processing Units (TPUs) can speedup standard AI tasks by more than two orders [28].

Parallel algorithms can be implemented on shared-memory or distributed-memory machines. Shared-memory based systems concern parallelizing one machine, whereas distributed-memory based systems involve several machines. Compared to distributed-memory based systems (as for example dCountAntom [6]) consisting of a massive amount of units, we rely on shared-memory based (used for instance in countAntom [5]) techniques, i.e., in particular plain consumer processors and graphics cards. Distributed units build on fast communication networks, and when designing such systems, the goal is to avoid communication overhead where possible to reduce the bottleneck induced by the transport channel. Shared-memory systems on the other hand – though limited by synchronization necessities – do not directly suffer from this issue and are in a sense incomparable to distributed-memory based systems. Consequently, we purposely focus on shared-memory based systems in this paper.

New Contribution

In this paper, we show that computationally involved problems such as #SAT or WMC benefit in practice from parallelization when the input instance has small treewidth. To this end, we implement the aforementioned dynamic programming approach for the first time on a GPU and provide an experimental evaluation. More specifically, our contributions are:

1. We engineer a novel architecture for GPU-based parameterized algorithms that allow for *parallel solving* of #SAT and WMC and where the runtime depends on the size of the computed decomposition of the graph representation of the formula. To this end, we traverse a tree decomposition similar to a sequential algorithm, but distribute the computation of tables among different computation units such that each potential row runs in one thread of the GPU, which is key for an efficient parallelization in practice.
2. We provide an OpenCL implementation gpusat[1] of two parameterized algorithms for the GPU. We highlight crucial algorithm engineering steps such as handling non-nice tree decompositions and specialized procedures that adjust the table sizes to the available number of computation units.
3. We provide rigorous experimental work where we consider an extensive number of dedicated #SAT and WMC instances and compare gpusat with a wide range of related solvers. We present upper bounds on the primal and incidence treewidth for our entire set of benchmark instances and compare the solving time with state-of-the-art solvers. In particular, our results show that gpusat is the fastest, *precise* solver for instances of treewidth up to 30 and is even able to solve certain instances of treewidth up to 45.

2 Solving #SAT by Dynamic Programming

Boolean Satisfiability and Weighted Model Counting

A literal is a Boolean variable x or its negation $\neg x$. A *clause* is a finite set of literals, interpreted as the disjunction of these literals. We say that a clause is *unit* if it is singleton. A *(CNF) formula* is a finite set of clauses, interpreted as the conjunction of its clauses. Let F be a formula. A *sub-formula* S of F consists of subsets of clauses of F. For a clause $c \in F$, $\text{var}(c)$ consists of all variables that occur in c and $\text{var}(F) := \bigcup_{c \in F} \text{var}(c)$. An *assignment* is a mapping $\alpha : \text{var}(F) \to \{0, 1\}$ and $2^{\text{var}(F)}$ the set of all assignments of F. $F(\alpha)$ is the formula F *under assignment* α obtained by removing all clauses c from F that contain a literal set to 1 by α and removing from the remaining clauses all literals set to 0 by α. An assignment α is *satisfying* if $F(\alpha) = \emptyset$. The problem #SAT asks to output the number of satisfying assignments of a formula. Let w be function that maps each literal of F to a real between 0 and 1. We call $w(\ell)$ the *weight* of literal ℓ. The *weight* of α is the product over the weights of its literals, i.e., $w(\alpha) := \Pi_{v \in \alpha^{-1}(1)} w(v) \cdot \Pi_{v \in \alpha^{-1}(0)} w(\neg v)$. The *weighted model count* of F is the sum of weights over all its satisfying assignments, i.e., $\Sigma_{\alpha \in 2^{\text{var}(F)}, F(\alpha) = \emptyset} w(\alpha)$. The problem WMC asks to output the weighted model count of F.

Tree Decomposition and Treewidth

A *tree decomposition (TD)* of a graph G is a pair $\mathcal{T} = (T, \chi)$ where T is a rooted tree (arborescence) and χ is a mapping that assigns to each node $t \in V(T)$ a set $\chi(t) \subseteq V(G)$, called a *bag*, such that the following conditions hold: (i) $V(G) = \bigcup_{t \in V(T)} \chi(t)$ and $E(G) \subseteq$

[1] Our solver is available at `github.com/daajoe/GPUSAT`.

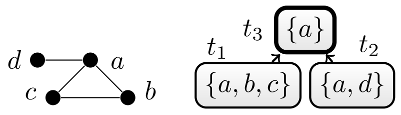


Figure 1 Primal graph P_F of F from Example 1 (left) with a tree decomposition $\mathcal{T}$ of the graph P_F (right).

$\bigcup_{t\in V(T)}\{\,\{u,v\} \mid u,v \in \chi(t)\,\}$; and (ii) for each $r,s,t \in T$, such that s lies on the path from r to t, we have $\chi(r) \cap \chi(t) \subseteq \chi(s)$. The *width* of $\mathcal{T}$, denoted width($\mathcal{T}$), is $\max_{t\in V(T)} |\chi(t)| - 1$. The *treewidth* tw($G$) of G is the minimum width($\mathcal{T}$) over all tree decompositions $\mathcal{T}$ of G. For arbitrary but fixed $w \geq 1$, it is feasible in linear time to decide if a graph has treewidth at most w and, if so, to compute a tree decomposition of width w [4]. Graphs that originate in the real-world often admit tree decompositions of small width [14]. Interestingly, one can use GPU-based implementations to compute the treewidth [46]. However, we use *htd* together with min-fill heuristics to compute TDs [1]. In that case, the width might not be minimal. In order to simplify cases in the theoretical algorithms, one uses for theoretical descriptions so-called nice TDs, which we can compute in linear time without increasing the width [29].

We need dedicated graph representations for satisfiability problems. The *primal graph* of a formula F has as vertices its variables and two variables are joined by an edge if they occur together in a clause of F. For a given node s of a TD (T,χ) of the primal graph of F, we let $F_s := \{\, c \mid c \in F, \text{var}(c) \subseteq \chi(s)\,\}$, i.e., clauses entirely covered by $\chi(s)$. The set $F_{\leq s}$ denotes the union over F_t for all descendant nodes $t \in V(T)$ of s. The *incidence graph* of a formula F is the bipartite graph on the clauses and variables of F, where a clause and a variable are joined by an edge if the variable occurs in the clause. We call the treewidth of the primal or incidence graph the *primal treewidth* or *incidence treewidth*, respectively.

▶ **Example 1.** Consider the formula $F := \{c_1 := a \vee b \vee \neg c, c_2 := \neg b \vee \neg a, c_3 := a \vee \neg d\}$. The primal graph P_F of formula F and a TD $\mathcal{T}$ of P_F are depicted in Figure 1. Intuitively, $\mathcal{T}$ allows to evaluate formula F in parts. Later when evaluating $F_{\leq t_3}$, we split into $F_{\leq t_1}$ and $F_{\leq t_2}$, which refer to $\{c_1, c_2\}$ and $\{c_3\}$, respectively.

Dynamic Programming on TDs

A #SAT or WMC solver based on *dynamic programming (DP)* evaluates the input formula F in parts along a given TD of F. For each node of the tree decomposition results are stored in *tables*. The algorithm works as outlined in Figure 2 and performs the following steps:

1. Construct a primal graph or incidence graph G of F.
2. Heuristically compute a tree decomposition (T,χ) of G.
3. DP: For every node t in post-order of $V(T)$, we run an algorithm $\mathbb{A} \in \{\mathbb{PRIM}, \mathbb{INC}\}$ that outputs a table τ_t and takes as input the node t, its bag $\chi(t)$, sub-formula F_t, and previously computed child tables C-Tabs of t (empty at the leaves).
4. Print the result by interpreting the table for root n of T.

We provide a brief intuition on $\mathbb{PRIM}$. For details and algorithm $\mathbb{PRIM}$ and $\mathbb{INC}$, we refer to the original source [40]. The main idea of $\mathbb{PRIM}$ is to store in table τ_t only assignments, which are restricted to bag χ_t depending on nice case distinctions of the node type, and its counters. From the count stored together with an assignment in the table at node t, we can read the number of satisfying assignments of the formula $F_{\leq t}$ for the induced sub-tree of T rooted at t. In the end, we can simply read the solution from the table at the root. $\mathbb{INC}$

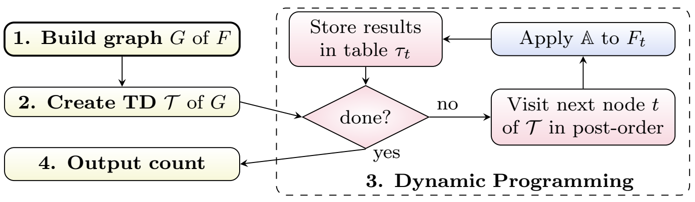

1. Build graph G of F
2. Create TD $\mathcal{T}$ of G
4. Output count
Store results in table τ_t
Apply $\mathbb{A}$ to F_t
done?
no
yes
Visit next node t of $\mathcal{T}$ in post-order
3. Dynamic Programming

Figure 2 Architecture of solvers based on dynamic programming on the CPU where an algorithm $\mathbb{A}$ modifies tables.

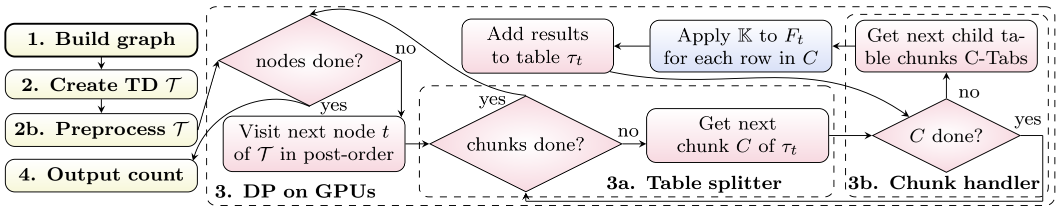

1. Build graph
2. Create TD $\mathcal{T}$
2b. Preprocess $\mathcal{T}$
4. Output count
nodes done?
no
yes
Visit next node t of $\mathcal{T}$ in post-order
3. DP on GPUs
Add results to table τ_t
Apply $\mathbb{K}$ to F_t for each row in C
Get next child table chunks C-Tabs
chunks done?
yes
no
Get next chunk C of τ_t
3a. Table splitter
C done?
no
yes
3b. Chunk handler

Figure 3 Architecture of our dynamic programming solver on the GPU where kernel $\mathbb{K}$ modifies each row individually and in parallel.

works similar, but requires more complex data structures. Both algorithms can be modified for computing the weighted model count.

3 GPU-based DP Architecture

Over the last decade there has been significant effort in the consumer market on graphics processing units (GPU) dedicated to render 3D graphics. GPUs are highly specialized in processing geometry and image information independent and in parallel. When one compares the actual computation power of such units to CPUs, GPUs are extremely cost efficient [28]. Recently, there is also increasingly strong interest in using such units for general purposes of parallelizable tasks in artificial intelligence and computation intensive applications such as number crunching [43].

In this section, we present an architecture for parallel dynamic programming on the GPU. In the dynamic programming algorithm, as outlined in Figure 2, nodes only depend on child nodes and in the table algorithm ($\mathbb{PRIM}$) rows in a table are entirely independent of each other. Consequently, there are two imminent ways to parallelize the execution. The first way is to compute tables for multiple nodes in parallel. This, however, does not allow for immediate massive parallelization due to dependencies to the child nodes. The second way is to distribute rows among different computation units. This allows with the right hindsight for massive parallelization, in particular, because the computation of a specific row is independent of any other row in the same table.

We would like to emphasize that the crucial tricks are (i) the way *how we parallelize* and (ii) a direct way to *represent potential assignments* (as explained below). Implementation techniques on the GPU and its parallelization follow a straight-forward programming paradigm and require in contrast to distributed-memory based systems [6] no parameter tuning.

The Kernel

Figure 3 outlines our dynamic programming approach on the GPU. It replaces Step 3 in the sequential dynamic programming approach above. The core of our solver is the procedure $\mathbb{K}$, which considers all possibly resulting rows at node t; even rows where the assignment in the row might not satisfy the sub-formula F_t, as we do not know the satisfiability in advance. For a node t, we call the table that consists of all possible rows *exhaustive table* (at node t). On the GPU a potential row can be seen as an output pixel that has to be computed. For all rows we take as common input the sub-formula F_t and specific to the row (assignment) corresponding rows in the tables of children of t. In terms of the methodology of programming on the GPU, procedure $\mathbb{K}$ is called a *kernel*. In our case, we spawn a (computation) thread at the GPU for each potential row of the exhaustive table with kernel $\mathbb{K}$. All threads have the same instructions $\mathbb{K}$, but start on different data. The underlying principle of the architecture is usually called single instruction multiple threads (SIMT). The kernel $\mathbb{K}$ depends on the type of the node just as before. For example, from the algorithm $\mathbb{PRIM}$ would still obtain several case distinctions but only for the different node types. In practice, however, we do not work on nice tree decompositions and therefore have case distinctions of mixed form. Further, it is crucial to tune our implementation towards simplicity and efficiency, which requires extensive *bit-twiddling* [2]. In particular, we need to reduce the number of execution paths, which we obtain by avoiding conditional jumps if possible. In other words, we prefer bit operations over if then else constructions to optimize for the underlying hardware. The GPU computation outputs counts of assignments that satisfy the sub-formula F_t. Processing all rows at once on the GPU allows us to compute the entire table in one GPU call, if the number of threads on the GPU and the available *video RAM (VRAM)* suffices, otherwise we run multiple “rounds” of computation.

Table Splitting

Even though running the kernel on the GPU allows us to obtain a parallel version of dynamic programming, our *main memory (RAM)* requirements are quite extensive and the required RAM might exceed the capacity of the VRAM on the GPU. Hence, we need to *split* large tables into smaller partitions of exhaustive tables (*chunks*). For a node t, this affects tables of the children of t as well as the exhaustive table at node t. We split the exhaustive table by a *table splitter* in Step 3a of Figure 3. A *chunk handler* then takes relevant chunks of the exhaustive table and spawns kernels depending on chunks of corresponding child tables as in Step 3b of Figure 3. The resulting counts for one exhaustive table chunk of this step are summed up accordingly and stored in table τ_t as previously explained.

TD Preprocessing

Orthogonally, in order to utilize the entire computation power of one cycle on the GPU, we merge several nodes of the tree decomposition into one node to obtain larger exhaustive tables. This reduces overhead caused by IO operations between the RAM and the VRAM and caused by spawning and deallocating GPU threads. Therefore, we run a *preprocessing* operation on the tree decomposition that merges small bags. This step may result in a tree decomposition that is not nice. Hence, we need to implement *more complex kernel* algorithms. Further, we obtain an even better GPU utilization by handling certain cases (introduce, remove, and leaf [40]) in one case and merging small bags, which share introduced and removed variables or clauses.

Data Types and Precision

In contrast to programs that are executed on the CPU, the instruction set for procedures on the GPU (kernels) is very limited and only a few data structures are available. In particular, there is no established data type for storing big numbers as directly offered in common programming languages [27, 48]. Still, we need dedicated data types to represent large numbers to express counts of satisfying assignments. Unfortunately, storing the exact number of solutions in each row of each table can be too expensive on the VRAM. Instead, we use the data type *double*. Hence, we cannot expect an exact solution when solving #SAT or WMC. However, we can use an extended type (*double4*) that combines four plain double types to increase the precision. Then, we can balance between a faster running time or higher precision. When solving #SAT we may run into a double or double4 overflow. Then, we can *relax* the instance into a weighted model count instance where all literals have the same weight, but less than 1, and reconstruct the original count at the end of the computation.

Implementation

We implemented our approach for dynamic programming on the GPU and kernels for the table algorithms $\mathbb{PRIM}$ and $\mathbb{INC}$ into our prototypical solver *gpusat*. We used OpenCL1.2 [37], which is a universal vendor and hardware independent computation framework, and C++11 for our implementation. Currently, we only use very limited formula preprocessing and simplifications during the search. Prior solving, we once propagate unit clauses in the usual way. If there is a table that does not contain any solution, we terminate and output that there is no solution. At a node t, we compute the sub-formula F_t using the CPU and start one GPU thread for each possible assignment. Kernels are compiled only once. The assignment is tied to the memory address, which then requires only memory for counts on the VRAM. We statically split tables based on the available memory on the GPU. We merge bags of small size as long as we obtain at most 14 variables in one bag.

4 Experimental Results

We performed an extensive series of experiments using several benchmark sets among them instances that originate in model counting and weighted model counting questions. All benchmarks as well as detailed results including raw data are publicly available[2]. Theoretically, we *do not* expect to solve formulas with graph representations of high treewidth. Therefore, we restricted the sets to instances where we were able to find tree decompositions of width below 30 using standard heuristic decomposers [1]. Nonetheless, we provide upper bounds on the treewidth for all instances of our benchmark sets. Since our benchmarks require entirely different type of hardware, we can only use wall clock time as a time measurement. Note that we used cheap consumer hardware for gpusat; whereas we used a very recent server hardware configuration for all other solvers.

Hardware

Our results were gathered on Ubuntu 16.04 LTS Linux machines kernel 4.4.0-101 and 4.14.0-041400, respectively, both pre-Spectre and pre-Meltdown kernels[3]. We ran non-GPU solvers on a cluster of 9 nodes. Each node is equipped with two Intel Xeon E5-2650 CPUs consisting of 12 physical cores each at 2.2 GHz clock speed and 256 GB RAM. Hyper threading was disabled. For gpusat we used a machine equipped with a consumer GPU: Intel Core i3-3245 CPU operating at 3.4 GHz, 16 GB RAM, and one Sapphire Pulse ITX Radeon RX 570 GPU

[2] See: Benchmark repository (including used tree decompositions) [19] and results/raw data [20].
[3] See: `spectreattack.com`

Table 1 Overview on upper bounds of the primal treewidth for considered benchmarks. # represents counting and W represents weighted model counting, number N of instances, number n of variables, median t Mdn of the runtime in seconds, maximum runtime t, and median Mdn and percentiles of the upper bounds on the treewidth.

set		origin	N	n Mdn	t[s] Mdn	(max.)	Mdn	50%	80%	95%
W	*Dqmr*	`Cachet`	660	140	0.0	(1.6)	28	28	42	44
W	*Grid*	`Cachet`	420	1825	0.2	(1.3)	29	29	39	71
W	*Plan*	`Cachet`	11	812	2.9	(9.3)	73	85	399	na
#	*Mixed*	`c2d`	14	1287	3.8	(15.9)	57	63	399	540
#	*Basic*	`fre/meel`	92	604	1.0	(9.3)	26	37	64	352
#	*Proj.*	`fre/meel`	308	62586	120.3	(880.4)	273	328	1084	na
#	*Weig.*	`fre/meel`	1080	200	0.1	(1.6)	28	28	40	48

running at 1.24 GHz with 32 compute units, 2048 shader units, and 4GB VRAM using driver amdgpu-pro 17.10.

Solvers

We benchmarked c2d [12], d4 [33], DSHARP [35], miniC2D [36], cnf2eadt [30], bdd_minisat_all [45], and sdd [13], which are based on knowledge compilation techniques. We also included recent approximate solvers ApproxMC [7] and sts [17], as well as pure CDCL-based solvers Clasp [25], Cachet [41], sharpCDCL[4] and sharpSAT [44]. Further, we considered the recent multi-core solver countAntom [5] utilizing exclusively all 12 physical cores, and DP based solvers on tree decompositions from related domains that allow with slight modifications for #SAT solving, i.e., dynasp [18] and dynQBF 1.1.1 [9]. We used all solvers with default options and ran gpusat with uniform weights 0.78 for #SAT experiments. All solvers allow for #SAT solving and sts, gpusat, miniC2D, and Cachet in addition support WMC[5].

Setup and Limits

In order to draw conclusions about the efficiency of gpusat, we mainly inspected the wall clock time *including decomposition time* and number of timeouts. We set a timeout of 900 seconds and limited available RAM to 8 GB per instance. For each instance we only used one tree decomposition, which was obtained by setting a random seed for the decomposer. All the tree decompositions together with the experimental data are provided as well[2]. Note that we avoid IO access on the CPU solvers whenever possible, i.e., we extract instances into the RAM before starting solving.

Benchmark Instances

We considered a selection of 2585 instances from various publicly available benchmark sets for model counting and weighted model counting, consisting of `Cachet` benchmarks[6] (1091 instances), `fre/meel` benchmarks[7](1451 instances), and `c2d` benchmarks[8] (14 instances).

[4] See: `tools.computational-logic.org`
[5] Note that in principle using a d-DNNF reasoner one can also use c2d and d4 to solve WMC.
[6] See: `cs.rochester.edu/u/kautz/Cachet`
[7] See: `tinyurl.com/countingbenchmarks`
[8] See: `reasoning.cs.ucla.edu/c2d`

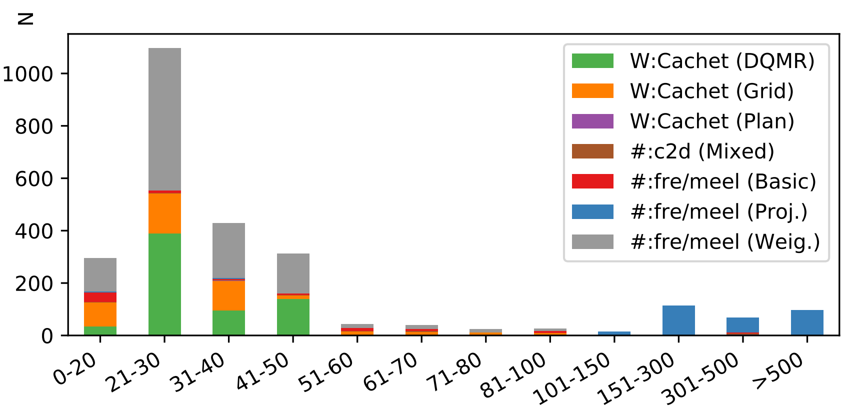


Figure 4 Distribution of instances in upper bound intervals on the primal treewidth over our benchmarks. The x-axis labels the intervals. The y-axis labels the number of observed instances.

Table 2 Number of WMC instances solved. Intervals are given with respect to primal graph. abs err indicates the absolute error. best indicates the number of instances the solver solved the fastest. † absolute weighted model counts were rounded to 3 decimal places. * indicates a significant (≥ 0.2 on average) absolute error.

solver	abs err	0-20	21-30	31-40	41-50	51-60	>60	best	$\sum$
Cachet	**0.0**	92	448	*108*	*105*	*2*	**9**	**476**	764
gpusat(i)	±**0.0**	*127*	487	83	101	0	0	42	798
gpusat(i4)	±**0.0**	*127*	432	75	90	0	0	0	724
gpusat(p)	±**0.0**	**128**	**526**	88	104	0	0	296	*846*
gpusat(p4)	±**0.0**	127	478	80	96	0	0	0	781
miniC2D	†±**0.0**	126	*513*	**143**	**110**	**5**	*6*	143	**903**
sts*	†±0.2	121	**533**	**200**	**152**	1	*6*	*na	***1013**

Treewidth

We computed upper bounds on the primal and incidence treewidth for our benchmarks. The sets contain instances that have the same graph representation. Upper bounds on the treewidth and running times to obtain a decomposition were quite similar for both the primal graph and the incidence graph, except for instances of the set *Proj*. Hence, we focus on an upper bound of the treewidth of the primal graph only and state them in intervals. Table 1 provides statistics on the benchmarks, including runtime of the decomposer to obtain a decomposition. Further, the decomposer ran 0.034s in median (max 1.57s) for instances of width 0–30, 0.132s (max 2.503s) for instances of width 31–40, and 0.054s (max 900.0s) over all instances. The decomposer did not output a decomposition within 900 seconds for 41 instances. Table 1 also states the median of the width of the obtained decompositions and its percentiles, which is the width below a given percentage the instances have. When considering the set *Dqmr*, even 99% of the instances have treewidth below 45. In contrast, the decomposer outputted only decompositions of very high width for instances from the set *Proj*. Figure 4 illustrates the distribution of number of instances (y-axis) and their respective upper bounds (x-axis) for primal treewidth. Considering all sets 54% of the instances have primal treewidth below 30, 70% of the instances have treewidth below 40, and 88% of the instances have treewidth below 150, and for 1% of the instances we obtained no result within the limit.

Table 3 Number of counting instances solved by sum of the top ten counting solvers and gpusat. The symbol * indicates that this gpusat configuration was not among the top ten.

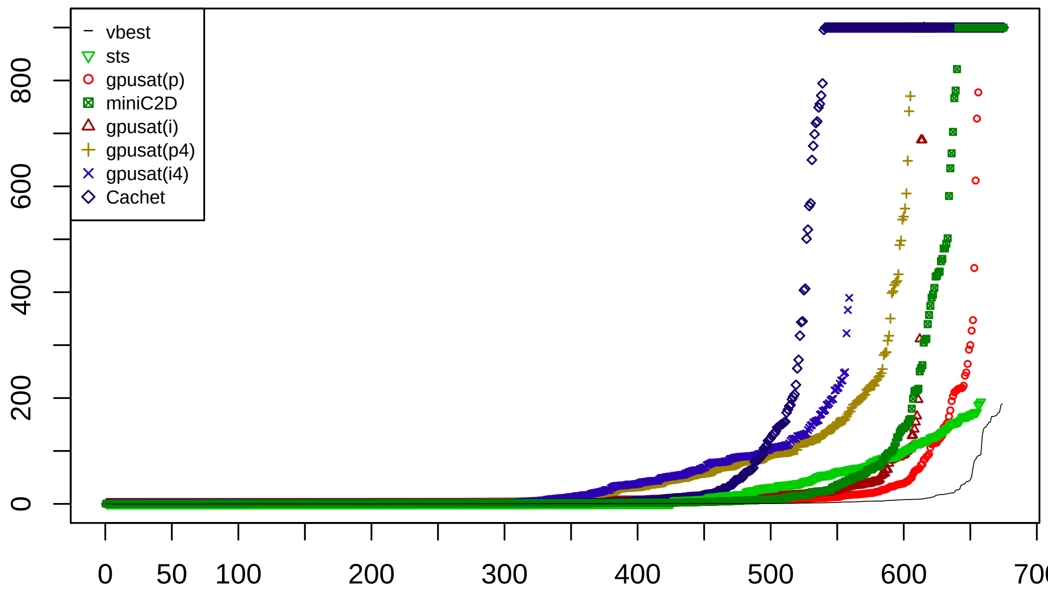
solver	0-20	21-30	31-40	41-50	51-60	>60	best	$\sum$
c2d	164	*519*	**175**	116	*20*	118	120	*1112*
Cachet	133	421	91	109	8	58	13	820
d4	**169**	510	*156*	*119*	**23**	**162**	191	**1139**
gpusat(i)	**169**	490	79	97	0	0	1	835
gpusat(i4)	168	427	70	89	0	0	1	*761
gpusat(p)	**169**	**523**	79	104	0	0	88	875
gpusat(p4)	**169**	478	79	97	0	0	0	823
miniC2D	*167*	491	137	103	8	67	2	973
sharpSAT	136	465	136	112	11	*124*	**483**	984
sts	162	448	101	**146**	10	45	*252*	912

Solved Instances, Runtime, and Error (WMC)

Table 2 gives an overview on the number of solved instances for weighted model counting benchmarks (`Cachet`) and the average error on the weighted model count of the solver. The *absolute error* is the difference of the weighted model count of the solver and the one obtained by Cachet. The configuration gpusat(p) and gpusat(i) refer to the primal and incidence graph implementation, respectively. gpusat(i4) or gpusat(p4) indicates that this configuration uses extended data type precision (double4). gpusat(p) solved the most instances in interval 0–20 and second most instances in interval 21–31 on benchmark sets for WMC; in interval 0–30 gpusat(p) solved the same number of instances as sts. However, gpusat(p) produced almost no absolute error on average ($\pm 1.42 \cdot 10^{-5}$). sts produced a very high absolute error on average ($\pm$ 0.2, stdev 0.8; avg relative error 1037) and had a relative error of more than one order on 56 instances (even when rounding weighted model counts to 3 decimal places). For example, sts outputted a weighted model count of 1.5 (0.873 Cachet) on instance *90-12-3-q.cnf* and 0.316 (0.001 Cachet) on instance *or-50-5-4-UC-20.cnf*. Slightly increasing the number of sampling iterations and samples per level resulted in slower runtimes than gpusat at similar error. Considering all instances gpusat(p) still solved the second most instances at sufficiently high accuracy. The double4 precision versions solved 65 and 74 less instances at negligible accuracy improvement, both versions provide at least the precision that Cachet offers. Figure 5 (top) illustrates runtime results on weighted model counting instances of width between 0 and 30 as cactus plot. When we directly compare gpusat(i) and gpusat(p), gpusat(i) solved 18 instances, which could not be solved by gpusat(p), and 70 instances vice versa. gpusat(i) was on 120 instances faster than gpusat(p) and 815 vice versa.

Solved Instances, Runtime, and Error (#SAT)

Table 3 gives an overview on the number of solved counting instances. gpusat(p) solved the most instances in interval 0–30. Considering all instances gpusat(p) solved the sixth most instances and surprisingly many instances in the interval 31–50. The double4 precision versions solved 52 (p) and 74 (i) less instances. In our experiments we observed on average an error of $4 \cdot 10^{-13}$ for double and $2 \cdot 10^{-32}$ for double4 when comparing to sharpSAT. Hence, we consider the precision error negligible. Without using a uniform weight for gpusat, we ran 80 (p) and 56 (i) times into a double overflow at similar runtime. Figure 5 (bottom) illustrates runtime results (in seconds) on instances of interval 0–30 as cactus plot.



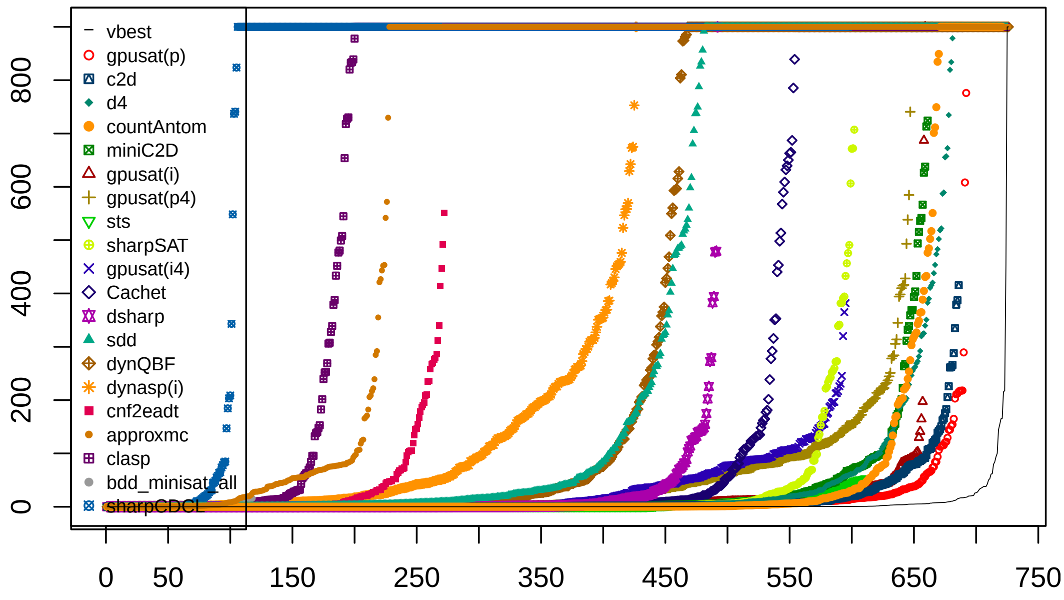


Figure 5 (Top): Runtime on WMC instances (`Cachet`) of primal treewidth at most 30 as cactus plot. (Bottom): Runtime on counting instances (`c2d`, `fre/meel`) of primal treewidth at most 30 as a cactus plot. vbest refers to the virtual best solver, i.e., the best runtime result among all solvers. The x-axis labels consecutive integers that identify instances. The instances are ordered by running time, individually for each solver. Hence, the figure does not provide insights on the solving time of the individual instances and solvers might solve instances fast, which is usually indicated by the virtual best solver. The y-axis labels the runtime (in seconds).

Runtime deviation

We tested gpusat with five different TDs (computed via *htd* [1]) to draw conclusions about runtime stability. The results indicate that the best, the average, and the median among those five tree decomposition still yield good runtime results. Regarding the number of tested instances it is practically quite unlikely to obtain the worst case behavior.

Discussion and Summary

Our results on upper bounds of the primal and incidence treewidth of WMC and #SAT benchmark instances, show that more than half of the instances have treewidth below 30 and more than two third have treewidth below 40. We observed that table splitting was necessary at width above 26. Since gpusat solved the vast majority of the instances in interval 0–30 (only 22 of the 670 WMC instances and 23 of the 721 #SAT instances were not solved), gpusat is highly suitable for the majority of the instances. It turns out that instances in interval 30–40 are still in reach for our solver, even certain instances of width upper bound 45 were solved. Overall gpusat was the fastest virtually exact solver in interval 0–30 for considered WMC and #SAT instances. Our results show that gpusat(p) solves more instances than gpusat(i) and instances often faster, which indicates that gpusat(p) benefits from its simpler algorithms. Using data types of higher precision does obviously not pay off. However, relaxing a #SAT instance into a WMC instance with uniform weights gives almost no precision loss. From our analysis, gpusat is not yet a general propose solver, but highly competitive if the treewidth is below 30. Since we can often find tree decompositions of small width in well below a second, it makes gpusat perfectly suitable for a portfolio approach.

5 Conclusion & Future Work

We introduced the OpenCL-based solver gpusat, which allows for solving #SAT and WMC using dynamic programming on tree decompositions running on consumer GPUs. Our solver parallelizes the computation of each table, vaguely speaking, a partial model count is represented by a pixel. Further, we provide insights on tuning parameterized algorithms for the GPU, including balancing VRAM utilization. We carried out rigorous experimental work, including establishing upper bounds for treewidth of commonly used benchmarks and comparing to most recent solvers. Our findings indicate that a majority of benchmark instances have treewidth below 30. Then, we can also heuristically compute tree decompositions in less than a second. Since gpusat is competitive on those instances, we show that implementations of parameterized algorithms on the GPU are a promising attempt to solve WMC. Hence, those algorithms are not just an interesting theoretical research direction, but its implementations are also competitive in practice. In our opinion, a wide range of applications [8, 15], even suggests to establish dedicated #SAT or WMC competitions, in particular, to obtain a wider picture on which method pays off for which domain.

The results of this paper give rise to several research questions. For instance, it would be interesting to determine the effect of formula preprocessing [32, 31] on the treewidth and solver runtimes. We conducted initial experiments, which suggest that preprocessors might drastically reduce the treewidth and hence increase the applicability of gpusat. Further, it might be fruitful to investigate on obtaining decompositions that have smaller width [21] or that are customized to improve efficiency of the dynamic programming algorithm [1]. An interesting further research direction is to study whether efficient data representation techniques can be combined with dynamic programming similar to techniques for QBF [9] and even be run in parallel on the GPU. Concerning potential overflows of counters for counting-only problems, we aim at analyzing and implementing further improvements as for example storing logarithmic counters [23]. At the same time we want to elaborate on ways to provide high-precision counter (libraries). Finally, parameterized algorithmics suggests recent parameters similar to treewidth [39], which can however be arbitrarily smaller than treewidth. We also aim for implementing these algorithms in OpenCL.

References

1 Michael Abseher, Nysret Musliu, and Stefan Woltran. htd – a free, open-source framework for (customized) tree decompositions and beyond. In Domenico Salvagnin and Michele Lombardi, editors, *Proceedings of the 14th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR'17)*, volume 10335 of *Lecture Notes in Computer Science*, pages 376–386, Padova, Italy, jun 2017. Springer Verlag. doi:10.1007/978-3-319-59776-8_30.

2 Sean Eron Anderson. Bit twiddling hacks. https://graphics.stanford.edu/~seander/bithacks.html, 2009.

3 Sander Beckers, Gorik De Samblanx, Floris De Smedt, Toon Goedemé, Lars Struyf, and Joost Vennekens. Parallel hybrid SAT solving using OpenCL. In Nico Roos, Mark Winands, and Jos Uiterwijk, editors, *Proceedings of the 24th Benelux Conference on Artificial Intelligence (BNAIC'12)*, pages 11–18, Maastricht, The Netherlands, 2012. Maastricht University.

4 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

5 Jan Burchard, Tobias Schubert, and Bernd Becker. Laissez-faire caching for parallel #SAT solving. In Marijn Heule and Sean Weaver, editors, *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT'15)*, volume 9340 of *Lecture Notes in Computer Science*, pages 46–61, Austin, TX, USA, 2015. Springer Verlag. doi:10.1007/978-3-319-24318-4_5.

6 Jan Burchard, Tobias Schubert, and Bernd Becker. Distributed parallel #sat solving. In Bronis R. de Supinski, editor, *Proceedings of the 2016 IEEE International Conference on Cluster Computing (CLUSTER'16)*, pages 326–335, 2016. doi:10.1109/CLUSTER.2016.20.

7 Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. Distribution-aware sampling and weighted model counting for SAT. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI'14)*, pages 1722–1730, Québec City, QC, Canada, 2014. The AAAI Press.

8 Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. Improving approximate counting for probabilistic inference: From linear to logarithmic sat solver calls. In Subbarao Kambhampati, editor, *Proceedings of 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*, pages 3569–3576, New York City, NY, USA, jul 2016. The AAAI Press. URL: https://bitbucket.org/kuldeepmeel/approxmc.

9 Günther Charwat and Stefan Woltran. Dynamic programming-based QBF solving. In Florian Lonsing and Martina Seidl, editors, *Proceedings of the 4th International Workshop on Quantified Boolean Formulas (QBF'16)*, volume 1719, pages 27–40. CEUR Workshop Proceedings (CEUR-WS.org), 2016. co-located with 19th International Conference on Theory and Applications of Satisfiability Testing (SAT'16).

10 Arthur Choi, Guy Van den Broeck, and Adnan Darwiche. Tractable learning for structured probability spaces: A case study in learning preference distributions. In Qiang Yang, editor, *Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*. The AAAI Press, 2015.

11 Alessandro Dal Palu, Agostino Dovier, Andrea Formisano, and Enrico Pontelli. Cud@SAT: SAT solving on GPUs. *Journal of Experimental & Theoretical Artificial Intelligence*, 27(3), 2015.

12 Adnan Darwiche. New advances in compiling CNF to decomposable negation normal form. In Ramon López De Mántaras and Lorenza Saitta, editors, *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04)*, pages 318–322, Valencia, Spain, 2004. IOS Press.

13 Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 819–826, Barcelona, Catalonia, Spain, jul 2011. AAAI Press/IJCAI.

14 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The pace 2017 parameterized algorithms and computational experiments challenge: The second iteration. In Daniel Lokshtanov and Naomi Nishimura, editors, *Proceedings of the 12th International Symposium on Parameterized and Exact Computation (IPEC'17)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 30:1—-30:13. Dagstuhl Publishing, 2017. `doi:10.4230/LIPIcs.IPEC.2017.30`.

15 Carmel Domshlak and Jörg Hoffmann. Probabilistic planning via heuristic forward search and weighted model counting. *Journal of Artificial Intelligence Research*, 30, 2007. `doi:10.1613/jair.2289`.

16 Leonardo Dueñas-Osorio, Kuldeep S. Meel, Roger Paredes, and Moshe Y. Vardi. Counting-based reliability estimation for power-transmission grids. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI'17)*, pages 4488–4494, San Francisco, CA, USA, feb 2017. The AAAI Press.

17 Stefano Ermon, Carla P. Gomes, and Bart Selman. Uniform solution sampling using a constraint solver as an oracle. In Nando de Freitas and Kevin Murphy, editors, *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI'12)*, pages 255–264, Catalina Island, CA, USA, aug 2012. AUAI Press.

18 Johannes K. Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Answer set solving with bounded treewidth revisited. In Marcello Balduccini and Tomi Janhunen, editors, *Proceedings of the 14th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'17)*, volume 10377 of *Lecture Notes in Computer Science*, pages 132–145, Espoo, Finland, jul 2017. Springer Verlag. `doi:10.1007/978-3-319-61660-5_13`.

19 Johannes K. Fichte, Markus Hecher, Stefan Woltran, and Markus Zisser. A Benchmark Collection of #SAT Instances and Tree Decompositions (Benchmark Set), 2018. `doi:10.5281/zenodo.1299752`.

20 Johannes K. Fichte, Markus Hecher, Stefan Woltran, and Markus Zisser. Analyzed Benchmarks and Raw Data on Experiments for gpusat (Dataset), jun 2018. `doi:10.5281/zenodo.1299742`.

21 Johannes K. Fichte, Neha Lodha, and Stefan Szeider. Sat-based local improvement for finding tree decompositions of small width. In Serge Gaspers and Toby Walsh, editors, *Proceedings on the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT'17)*, pages 401–411, Melbourne, VIC, Australia, aug 2017. Springer Verlag. `doi:10.1007/978-3-319-66263-3_25`.

22 Ferdinando Fioretto, Enrico Pontelli, William Yeoh, and Rina Dechter. Accelerating exact and approximate inference for (distributed) discrete optimization with GPUs. *Constraints*, 23(1):1–23, 2017. `doi:10.1007/s10601-017-9274-1`.

23 Philippe Flajolet. Approximate counting: A detailed analysis. *BIT Numerical Mathematics*, 25(1):113–134, 1985. `doi:10.1007/BF01934993`.

24 Hironori Fujii and Noriyuki Fujimoto. Gpu acceleration of bcp procedure for sat algorithms. In Hamid R. Arabnia, Hiroshi Ishii, Minoru Ito Kazuki Joe, and Hiroaki Nishikawa, editors, *Proceedings of the 24th International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'12)*, pages 10–16, Las Vegas, NV, USA, 2012. CSREA Press.

25 Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187-188:52–89, 2012. `doi:10.1016/j.artint.2012.04.001`.

26 Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Chapter 20: Model counting. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 633–654. IOS Press, Amsterdam, Netherlands, 2009. `doi:10.3233/978-1-58603-929-5-633`.

27 Torbjörn Granlund, Gunnar Sjödin, Hans Riesel, Richard Stallman, Brian Beuning, Doug Lea, Paul Zimmermann, Ken Weber, Per Bothner, Joachim Hollman, Bennet Yee, Andreas Schwab, Robert Harley, David Seal, Torsten Ekedahl, Linus Nordberg, Kevin Ryde, Kent Boortz, Steve Root, Gerardo Ballabio, Jason Moxham, Niels Möller, Alberto Zanoni, Marco Bodrato, David Harvey, Martin Boij, Marc Glisse, David S Miller, Mark Sofroniou, and Ulrich Weigand. The GNU multiple precision arithmetic library. `https://gmplib.org`, 2016.

28 Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In David Brooks, editor, *Proceedings of the 44th International Symposium on Computer Architecture (ISCA'17)*, pages 1–12, Toronto, ON, Canada, jun 2017. `doi:10.1145/3079856.3080246`.

29 Ton Kloks. *Treewidth. Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer Verlag, 1994. `doi:10.1007/BFb0045375`.

30 Frédéric Koriche, Jean-Marie Lagniez, Pierre Marquis, and Samuel Thomas. Knowledge compilation for model counting: Affine decision trees. In Francesca Rossi and Sebastian Thrun, editors, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*, Beijing, China, aug 2013. The AAAI Press.

31 Jean-Marie Lagniez, Emmanuel Lonca, and Pierre Marquis. Improving model counting by leveraging definability. In Subbarao Kambhampati, editor, *Proceedings of 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*, pages 751–757, New York City, NY, USA, 2016. The AAAI Press.

32 Jean-Marie Lagniez and Pierre Marquis. Preprocessing for propositional model counting. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI'14)*, pages 2688–2694, Québec City, QC, Canada, 2014. The AAAI Press.

33 Jean-Marie Lagniez and Pierre Marquis. An improved decision-DDNF compiler. In Carles Sierra, editor, *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, pages 667–673, Melbourne, VIC, Australia, 2017. The AAAI Press.

34 Norbert Manthey. Towards next generation sequential and parallel SAT solvers. *KI - Kuenstliche Intelligenz*, 30(3-4):339–342, 2016. `doi:10.1007/s13218-015-0406-8`.

35 Sheila A. Muise, Christian J .and McIlraith, J. Christopher Beck, and Eric I. Hsu. Dsharp: Fast d-DNNF compilation with sharpSAT. In Leila Kosseim and Diana Inkpen, editors, *Proceedings of the 25th Canadian Conference on Artificial Intelligence (AI'17)*, volume

7310 of *Lecture Notes in Computer Science*, pages 356–361, Toronto, ON, Canada, 2012. Springer Verlag. doi:10.1007/978-3-642-30353-1_36.

36 Umut Oztok and Adnan Darwiche. A top-down compiler for sentential decision diagrams. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 3141–3148. The AAAI Press, 2015.

37 Jonathan Passerat-Palmbach and David Hill. *OpenCL: A suitable solution to simplify and unify high performance computing developments*, chapter 8. Saxe-Coburg Publications, 2013.

38 Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1–2), 1996. doi:10.1016/0004-3702(94)00092-1.

39 Sigve Hortemo Sæther, Jan Arne Telle, and Martin Vatshelle. Solving #SAT and MAXSAT by dynamic programming. *Journal of Artificial Intelligence Research*, 54:59–82, 2015.

40 Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *Journal of Discrete Algorithms*, 8(1):50—-64, 2010. doi:10.1016/j.jda.2009.06.002.

41 Tian Sang, Fahiem Bacchus, Paul Beame, Henry Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. In Holger H. Hoos and David G. Mitchell, editors, *Online Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, Vancouver, BC, Canada, 2004.

42 Tian Sang, Paul Beame, and Henry Kautz. Performing bayesian inference by weighted model counting. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings of the 29th National Conference on Artificial Intelligence (AAAI'05)*. The AAAI Press, 2005.

43 Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full sha-1. In Jonathan Katz and Hovav Shacham, editors, *Proceedings of the 37th Annual International Cryptology Conference (Advances in Cryptology – CRYPTO'17)*, volume 10401 of *Lecture Notes in Computer Science*, pages 570–596, Santa Barbara, CA, USA, 2017. Springer Verlag. doi:10.1007/978-3-319-63688-7_19.

44 Marc Thurley. sharpSAT – counting models with advanced component caching and implicit BCP. In Armin Biere and Carla P. Gomes, editors, *Proceedings of the 9th International Conference Theory and Applications of Satisfiability Testing (SAT'06)*, pages 424–429, Seattle, WA, USA, 2006. Springer Verlag. doi:10.1007/11814948_38.

45 Takahis Toda and Takehide Soh. Implementing efficient all solutions SAT solvers. *ACM Journal of Experimental Algorithmics*, 21:1.12, 2015. Special Issue SEA 2014, Regular Papers and Special Issue ALENEX 2013.

46 Tom C. van der Zanden and Hans L. Bodlaender. Computing treewidth on the GPU. In Daniel Lokshtanov and Naomi Nishimura, editors, *Proceedings of the 12th International Symposium on Parameterized and Exact Computation (IPEC'17)*, volume 89 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:13, Dagstuhl, Germany, 2018. Dagstuhl Publishing. doi:10.4230/LIPIcs.IPEC.2017.29.

47 Yexiang Xue, Arthur Choi, and Adnan Darwiche. Basing decisions on sentences in decision diagrams. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*, Toronto, ON, Canada, 2012. The AAAI Press.

48 Moshe Zadka and Guido van Rossum. PEP 237 – unifying long integers and integers. https://www.python.org/dev/peps/pep-0237/, 2001.

Light Spanners for High Dimensional Norms via Stochastic Decompositions

Arnold Filtser[1]
Ben-Gurion University of the Negev, Beer-Sheva, Israel
arnoldf@cs.bgu.ac.il

Ofer Neiman[2]
Ben-Gurion University of the Negev, Beer-Sheva, Israel
neimano@cs.bgu.ac.il

Abstract

Spanners for low dimensional spaces (e.g. Euclidean space of constant dimension, or doubling metrics) are well understood. This lies in contrast to the situation in high dimensional spaces, where except for the work of Har-Peled, Indyk and Sidiropoulos (SODA 2013), who showed that any n-point Euclidean metric has an $O(t)$-spanner with $\tilde{O}(n^{1+1/t^2})$ edges, little is known.

In this paper we study several aspects of spanners in high dimensional normed spaces. First, we build spanners for finite subsets of ℓ_p with $1 < p \leq 2$. Second, our construction yields a spanner which is both sparse and also *light*, i.e., its total weight is not much larger than that of the minimum spanning tree. In particular, we show that any n-point subset of ℓ_p for $1 < p \leq 2$ has an $O(t)$-spanner with $n^{1+\tilde{O}(1/t^p)}$ edges and lightness $n^{\tilde{O}(1/t^p)}$.

In fact, our results are more general, and they apply to any metric space admitting a certain low diameter stochastic decomposition. It is known that arbitrary metric spaces have an $O(t)$-spanner with lightness $O(n^{1/t})$. We exhibit the following tradeoff: metrics with decomposability parameter $\nu = \nu(t)$ admit an $O(t)$-spanner with lightness $\tilde{O}(\nu^{1/t})$. For example, n-point Euclidean metrics have $\nu \leq n^{1/t}$, metrics with doubling constant λ have $\nu \leq \lambda$, and graphs of genus g have $\nu \leq g$. While these families do admit a $(1+\epsilon)$-spanner, its lightness depend exponentially on the dimension (resp. $\log g$). Our construction alleviates this exponential dependency, at the cost of incurring larger stretch.

2012 ACM Subject Classification Theory of computation $\rightarrow$ Sparsification and spanners

Keywords and phrases Spanners, Stochastic Decompositions, High Dimensional Euclidean Space, Doubling Dimension, Genus Graphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.29

Acknowledgements We would like to thank an anonymous reviewer for useful comments.

1 Introduction

1.1 Spanners

Given a metric space (X, d_X), a weighted graph $H = (X, E)$ is a *t-spanner* of X, if for every pair of points $x, y \in X$, $d_X(x, y) \leq d_H(x, y) \leq t \cdot d_X(x, y)$ (where d_H is the shortest path metric in H). The factor t is called the *stretch* of the spanner. Two important parameters of interest are: the *sparsity* of the spanner, i.e. the number of edges, and the *lightness* of the

[1] Partially supported by the Lynn and William Frankel Center for Computer Sciences, ISF grant 1817/17, and by BSF Grant 2015813.
[2] Partially supported by ISF grant 1817/17, and by BSF Grant 2015813.

© Arnold Filtser and Ofer Neiman;
licensed under Creative Commons License CC-BY

spanner, which is the ratio between the total weight of the spanner and the weight of the minimum spanning tree (MST).

The tradeoff between stretch and sparsity/lightness of spanners is the focus of an intensive research effort, and low stretch spanners were used in a plethora of applications, to name a few: Efficient broadcast protocols [8, 9], network synchronization [6, 49, 8, 9, 48], data gathering and dissemination tasks [14, 60, 22], routing [61, 49, 50, 57], distance oracles and labeling schemes [47, 58, 53], and almost shortest paths [19, 52, 23, 25, 28].

Spanners for general metric spaces are well understood. The seminal paper of [4] showed that for any parameter $k \geq 1$, any metric admits a $(2k-1)$-spanner with $O(n^{1+1/k})$ edges, which is conjectured to be best possible. For light spanners, improving [17, 24], it was shown in [18] that for every constant $\epsilon > 0$ there is a $(2k-1)(1+\epsilon)$-spanner with lightness $O(n^{1/k})$ and at most $O(n^{1+1/k})$ edges.

There is an extensive study of spanners for restricted classes of metric spaces, most notably subsets of low dimensional Euclidean space, and more generally doubling metrics.[3] For such low dimensional metrics, much better spanners can be obtained. Specifically, for n points in d-dimensional Euclidean space, [54, 59, 21] showed that for any $\epsilon \in (0, \frac{1}{2})$ there is a $(1+\epsilon)$-spanner with $n \cdot \epsilon^{-O(d)}$ edges and lightness $\epsilon^{-O(d)}$ (further details on Euclidean spanners could be found in [45]). This result was recently generalized to doubling metrics by [12], with $\epsilon^{-O(\mathrm{ddim})}$ lightness and $n \cdot \epsilon^{-O(\mathrm{ddim})}$ edges (improving [55, 30, 29]). Such low stretch spanners were also devised for metrics arising from certain graph families. For instance, [4] showed that any planar graph admits a $(1+\epsilon)$-spanner with lightness $O(1/\epsilon)$. This was extended to graphs with small genus[4] by [31], who showed that every graph with genus $g > 0$ admits a spanner with stretch $(1+\epsilon)$ and lightness $O(g/\epsilon)$. A long sequence of works for other graph families, concluded recently with a result of [13], who showed $(1+\epsilon)$-spanners for graphs excluding K_r as a minor, with lightness $\approx O(r/\epsilon^3)$.

In all these results there is an exponential dependence on a certain parameter of the input metric space (the dimension, the logarithm of the genus/minor-size), which is unfortunately unavoidable for small stretch (for all n-point metric spaces the dimension/parameter is at most $O(\log n)$, while spanner with stretch better than 3 requires in general $\Omega(n^2)$ edges [58]). So when the relevant parameter is small, light spanners could be constructed with stretch arbitrarily close to 1. However, in metrics arising from actual data, the parameter of interest may be moderately large, and it is not known how to construct light spanners avoiding the exponential dependence on it. In this paper, we devise a tradeoff between stretch and sparsity/lightness that can diminish this exponential dependence. To the best of our knowledge, the only such tradeoff is the recent work of [34], who showed that n-point subsets of Euclidean space (in any dimension) admit a $O(t)$-spanner with $\tilde{O}(n^{1+1/t^2})$ edges (without any bound on the lightness).

1.2 Stochastic Decompositions

In a (stochastic) decomposition of a metric space, the goal is to find a partition of the points into clusters of low diameter, such that the probability of nearby points to fall into different clusters is small. More formally, for a metric space (X, d_X) and parameters $t \geq 1$

[3] A metric space (X, d) has doubling constant λ if for every $x \in X$ and radius $r > 0$, the ball $B(x, 2r)$ can be covered by λ balls of radius r. The doubling dimension is defined as $\mathrm{ddim} = \log_2 \lambda$. A d-dimensional ℓ_p space has $\mathrm{ddim} = \Theta(d)$, and every n point metric has $\mathrm{ddim} = O(\log n)$.

[4] The *genus* of a graph is minimal integer g, such that the graph could be drawn on a surface with g "handles".

and $\delta = \delta(|X|, t) \in [0, 1]$, we say that the metric is (t, δ)-decomposable, if for every $\Delta > 0$ there is a probability distribution over partitions of X into clusters of diameter at most $t \cdot \Delta$, such that every two points of distance at most Δ have probability at least δ to be in the same cluster.

Such decompositions were introduced in the setting of distributed computing [7, 43], and have played a major role in the theory of metric embedding [10, 51, 26, 38, 39, 1], distance oracles and routing [44, 2], multi-commodity flow/sparsest cut gaps [41, 37] and also were used in approximation algorithms and spectral methods [15, 36, 11]. We are not aware of any direct connection of these decompositions to spanners (except spanners for general metrics implicit in [44, 2]).

Note that our definition is slightly different than the standard one. The probability δ that a pair $x, y \in X$ is in the same cluster may depend on $|X|$ and t, but unlike previous definitions, it does not depend on the precise value of $d_X(x, y)$ (rather, only on the fact that it is bounded by Δ). This simplification suits our needs, and it enables us to capture more succinctly the situation for high dimensional normed spaces, where the dependence of δ on $d_X(x, y)$ is non-linear. These stochastic decompositions are somewhat similar to Locality Sensitive Hashing (LSH), that were used by [34] to construct spanners. The main difference is that in LSH, far away points may be mapped to the same cluster with some small probability, and more focus was given to efficient computation of the hash function. It is implicit in [34] that existence of good LSH imply sparse spanners.

A classic tool for constructing spanners in normed and doubling spaces is WSPD (Well Separated Pair Decomposition, see [16, 56, 35]). Given a set of points P, a WSPD is a set of pairs $\{(A_i, B_i)\}_i$ of subsets of P, where the diameters of A_i and B_i are at most an ϵ-fraction of $d(A_i, B_i)$, and such that for every pair $x, y \in P$ there is some i with $(x, y) \in A_i \times B_i$. A WSPD is designed to create a $(1 + O(\epsilon))$-spanner, by adding an arbitrary edge between a point in A_i and a point in B_i for every i (as opposed to our construction, based on stochastic decompositions, in which we added only inner-cluster edges). An exponential dependence on the dimension is unavoidable with such a low stretch, thus it is not clear whether one can use a WSPD to obtain very sparse or light spanners in high dimensions.

1.3 Our Results

Our main result is exhibiting a connection between stochastic decompositions of metric spaces, and light spanners. Specifically, we show that if an n-point metric is (t, δ)-decomposable, then for any constant $\epsilon > 0$, it admits a $(2 + \epsilon) \cdot t$-spanner with $\tilde{O}(n/\delta)$ edges and lightness $\tilde{O}(1/\delta)$. (Abusing notation, $\tilde{O}$ hides polylog(n) factors.)

It can be shown that Euclidean metrics are $(t, n^{-O(1/t^2)})$-decomposable, thus our results extends [34] by providing a smaller stretch $(2 + \epsilon) \cdot t$-spanner, which is both sparse – with $\tilde{O}(n^{1+O(1/t^2)})$ edges – and has lightness $\tilde{O}(n^{O(1/t^2)})$. For d-dimensional Euclidean space, where $d = o(\log n)$ we can obtain lightness $\tilde{O}(2^{O(d/t^2)})$ and $\tilde{O}(n \cdot 2^{O(d/t^2)})$ edges. We also show that n-point subsets of ℓ_p spaces for any fixed $1 < p < 2$ are $(t, n^{-O(\log^2 t/t^p)})$-decomposable, which yields light spanners for such metrics as well.

In addition, metrics with doubling constant λ are $(t, \lambda^{-O(1/t)})$-decomposable [33, 1], and graphs with genus g are $(t, g^{-O(1/t)})$-decomposable [40, 3], which enables us to alleviate the exponential dependence on ddim and $\log g$ in the sparsity/lightness by increasing the stretch. See Table 1 for more details. (We remark that for graphs excluding K_r as a minor, the current best decomposition achieves probability only $2^{-O(r/t)}$ [3]; if this will be improved to the conjectured $r^{-O(1/t)}$, then our results would provide interesting spanners for this family as well.)

Table 1 In this table we summarize some corollaries of our main result. The metric spaces have cardinality n, and $\tilde{O}$ hides (mild) polylog(n) factors. The stretch t is a parameter ranging between 1 and $\log n$.

	Stretch	**Lightness**	**Sparsity**	
Euclidean space	$O(t)$	$\tilde{O}(n^{1/t^2})$	$\tilde{O}(n^{1+1/t^2})$	Corollary 6
	$O(\sqrt{\log n})$	$\tilde{O}(1)$	$\tilde{O}(n)$	
ℓ_p space, $1 < p < 2$	$O(t)$	$\tilde{O}(n^{\log^2 t/t^p})$	$\tilde{O}(n^{1+\log^2 t/t^p})$	Corollary 7
	$O((\log n \cdot \log\log n)^{1/p})$	$\tilde{O}(1)$	$\tilde{O}(n)$	
Doubling constant λ	$O(t)$	$\tilde{O}(\lambda^{1/t})$	$\tilde{O}(n \cdot \lambda^{1/t})$	Corollary 8
	$O(\log \lambda)$	$\tilde{O}(1)$	$\tilde{O}(n)$	
Graph with genus g	$O(t)$	$\tilde{O}(g^{1/t})$	$O(n+g)$	Corollary 9
	$O(\log g)$	$\tilde{O}(1)$	$O(n+g)$	

Note that up to polylog(n) factors, our stretch-lightness tradeoff generalizes the [18] spanner for general metrics, which has stretch $(2t-1)(1+\epsilon)$ and lightness $O(n^{1/t})$. Define for a (t,δ)-decomposable metric the parameter $\nu = 1/\delta^t$. Then we devise for such a metric a $(2t-1)(1+\epsilon)$-spanner with lightness $O(\nu^{1/t})$.

For example, consider an n-point metric with doubling constant $\lambda = 2^{\sqrt{\log n}}$. No spanner with stretch $o(\log n/\log\log n)$ and lightness $\tilde{O}(1)$ for such a metric was known. Our result implies such a spanner, with stretch $O(\sqrt{\log n})$.

We also remark that the existence of light spanners does not imply decomposability. For example, consider the shortest path metrics induced by bounded-degree expander graphs. Even though these metrics have the (asymptotically) worst possible decomposability parameters (they are only $(t, n^{-\Omega(1/t)})$-decomposable [42]), they nevertheless admit 1-spanners with constant lightness (the spanner being the expander graph itself).

2 Preliminaries

Given a metric space (X, d_X), let T denote its minimum spanning tree (MST) of weight L. For a set $A \subseteq X$, the diameter of A is $\text{diam}(A) = \max_{x,y\in A} d_X(x,y)$. Assume, as we may, that the minimal distance in X is 1.

By O_ϵ we denote asymptotic notation which hides polynomial factors of $\frac{1}{\epsilon}$, that is $O_\epsilon(f) = O(f)\cdot\text{poly}(\frac{1}{\epsilon})$. Unless explicitly specified otherwise, all logarithms are in base 2.

Nets. For $r > 0$, a set $N \subseteq X$ is an *r-net*, if (1) for every $x \in X$ there is a point $y \in N$ with $d_X(x,y) \le r$, and (2) every pair of net points $y, z \in N$ satisfy $d_X(y,z) > r$. It is well known that nets can be constructed in a greedy manner. For $0 < r_1 \le r_2 \le \cdots \le r_s$, a *hierarchical net* is a collection of nested sets $X \supseteq N_1 \supseteq N_2 \supseteq \cdots \supseteq N_s$, where each N_i is an r_i-net. Since N_{i+1} satisfies the second condition of a net with respect to radius r_i, one can obtain N_i from N_{i+1} by greedily adding points until the first condition is satisfied as well. In the following claim we argue that nets are sparse sets with respect to the MST weight.

▶ **Claim 1.** *Consider a metric space (X, d_X) with MST of weight L, let N be an r-net, then $|N| \le \frac{2L}{r}$.*

Proof. Let T be the MST of X, note that for every $x, y \in N$, $d_T(x,y) \ge d_X(x,y) > r$. For a point $x \in N$, $B_T(x,b) = \{y \in X \mid d_T(x,y) \le b\}$ is the ball of radius b around x in the MST metric. We say that an edge $\{y,z\}$ of T is *cut* by the ball $B_T(x,b)$ if $d_T(x,y) < b < d_T(x,z)$.

Consider the set $\mathcal{B}$ of balls of radius $r/2$ around the points of N. We can subdivide[5] the edges of T until no edge is cut by any of the balls of $\mathcal{B}$. Note that the subdivisions do not change the total weight of T nor the distances between the original points of X.

If both the endpoints of an edge e belong to the ball B, we say that the edge e is internal to B. By the second property of nets, and since $B_T(x,b) \subseteq B_X(x,b)$, the set of internal edges corresponding to the balls $\mathcal{B}$ are disjoint. On the other hand, as the tree is connected, the weight of the internal edges in each ball must be at least $r/2$. As the total weight is bounded by L, the claim follows. ◀

Stochastic Decompositions. Consider a *partition* $\mathcal{P}$ of X into disjoint clusters. For $x \in X$, we denote by $\mathcal{P}(x)$ the cluster $P \in \mathcal{P}$ that contains x. A partition $\mathcal{P}$ is Δ-*bounded* if for every $P \in \mathcal{P}$, $\mathrm{diam}(P) \le \Delta$. If a pair of points x,y belong to the same cluster, i.e. $\mathcal{P}(x) = \mathcal{P}(y)$, we say that they are *clustered* together by $\mathcal{P}$.

▶ **Definition 2.** For metric space (X, d_X) and parameters $t \ge 1$, $\Delta > 0$ and $\delta \in [0,1]$, a distribution $\mathcal{D}$ over partitions of X is called a (t,Δ,δ)-decomposition, if it fulfills the following properties.
- Every $\mathcal{P} \in \mathrm{supp}(\mathcal{D})$ is $t\cdot\Delta$-bounded.
- For every $x,y \in X$ such that $d_X(x,y) \le \Delta$, $\Pr_{\mathcal{D}}[\mathcal{P}(x) = \mathcal{P}(y)] \ge \delta$.

A metric is (t,δ)-decomposable, where $\delta = \delta(|X|,t)$, if it admits a (t,Δ,δ)-decomposition for any $\Delta > 0$. A family of metrics is (t,δ)-decomposable if each member (X,d_X) in the family is (t,δ)-decomposable.

We observe that if a metric (X,d_X) is $(t,\delta(|X|,t))$-decomposable, then also every sub-metric $Y \subseteq X$ is $(t,\delta(|X|,t))$-decomposable. In some cases Y is also $(t,\delta(|Y|,t))$-decomposable (we will exploit these improved decompositions for subsets of ℓ_p). The following claim argues that sampling $O(\frac{\log n}{\delta})$ partitions suffices to guarantee that every pair is clustered at least once.

▶ **Claim 3.** *Let (X,d_X) be a metric space which admits a (t,Δ,δ)-decomposition, and let $N \subseteq X$ be of size $|N| = n$. Then there is a set $\{\mathcal{P}_1,\dots,\mathcal{P}_\varphi\}$ of $t\cdot\Delta$-bounded partitions of N, where $\varphi = \frac{2\ln n}{\delta}$, such that every pair $x,y \in N$ at distance at most Δ is clustered together by at least one of the $\mathcal{P}_i$.*

Proof. Let $\{\mathcal{P}_1,\dots,\mathcal{P}_\varphi\}$ be i.i.d partitions drawn from the (t,Δ,δ)-decomposition of X. Consider a pair $x,y \in N$ at distance at most Δ. The probability that x,y are not clustered in any of the partitions is bounded by

$$\Pr[\forall i,\quad \mathcal{P}_i(x) \neq \mathcal{P}_i(y)] \ \le\ (1-\delta)^{(2\ln n)/\delta} \ \le\ \frac{1}{n^2}\ .$$

The claim now follows by the union bound. ◀

3 Light Spanner Construction

In this section we present a generalized version of the algorithm of [34], depicted in Algorithm 1. The differences in execution and analysis are: (1) Our construction applies to general decomposable metric spaces – we use decompositions rather than LSH schemes. (2) We

[5] To subdivide an edge $e = \{x,y\}$ of weight w the following steps are taken: (1) Delete the edge e. (2) Add a new vertex v_e. (3) Add two new edges $\{x,v_e\}, \{v_e,y\}$ with weights $\alpha\cdot w$ and $(1-\alpha)\cdot w$ for some $\alpha \in (0,1)$.

Algorithm 1 $H =$ `Spanner-From-Decompositions`$((X, d_X), t, \epsilon)$.

```
1: Let $N_0 \supseteq N_1 \supseteq \cdots \supseteq N_{\log_{1+\epsilon} L}$ be a hierarchical net, where $N_i$ is $\epsilon \cdot \Delta_i = \epsilon \cdot (1+\epsilon)^i$-net of $(X, d_X)$.
2: for $i \in \{0, 1, \ldots, \log_{1+\epsilon} L\}$ do
3:     For parameters $\Delta = (1+2\epsilon)\Delta_i$ and $t$, let $\mathcal{P}_1, \ldots, \mathcal{P}_{\varphi_i}$ be the set of $t \cdot \Delta$-bounded partitions guaranteed by Claim 3 on the set $N_i$.
4:     for $j \in \{1, \ldots, \varphi_i\}$ and $P \in \mathcal{P}_j$ do
5:         Let $v_P \in P$ be an arbitrarily point.
6:         Add to $H$ an edge from every point $x \in P \setminus \{v_P\}$ to $v_P$.
7:     end for
8: end for
9: return $H$.
```

analyze the lightness of the resulting spanners. (3) We achieve stretch $t \cdot (2+\epsilon)$ rather than $O(t)$.

The basic idea is as follows. For every weight scale $\Delta_i = (1+\epsilon)^i$, construct a sequence of $t \cdot \Delta_i$-bounded partitions $\mathcal{P}_1, \ldots, \mathcal{P}_\varphi$ such that every pair x, y at distance $\leq \Delta_i$ will be clustered together at least once. Then, for each $j \in [\varphi]$ and every cluster $P \in \mathcal{P}_j$, we pick an arbitrary root vertex $v_P \in P$, and add to our spanner edges from v_P to all the points in P. This ensures stretch $2t \cdot (1+\epsilon)$ for all pairs with $d_X(x,y) \in [(1-\epsilon)\Delta_i, \Delta_i]$. Thus, repeating this procedure on all scales $i = 1, 2, \ldots$ provides a spanner with stretch $2t \cdot (1+\epsilon)$.

However, the weight of the spanner described above is unbounded. In order to address this problem at scale Δ_i, instead of taking the partitions over all points, we partition only the points of an $\epsilon\Delta_i$-net. The stretch is still small: x, y at distance Δ_i will have nearby net points $\tilde{x}, \tilde{y}$. Then, a combination of newly added edges with older ones will produce a short path between x to y. The bound on the lightness will follow from the observation that the number of net points is bounded with respect to the MST weight.

▶ **Theorem 4.** *Let (X, d_X) be a (t, δ)-decomposable n-point metric space. Then for every $\epsilon \in (0, 1/8)$, there is a $t \cdot (2+\epsilon)$-spanner for X with lightness $O_\epsilon\left(\frac{t}{\delta} \cdot \log^2 n\right)$ and $O_\epsilon\left(\frac{n}{\delta} \cdot \log n \cdot \log t\right)$ edges.*

Proof. We will prove stretch $t \cdot (2 + O(\epsilon))$ instead of $t \cdot (2+\epsilon)$. This is good enough, as post factum we can scale ϵ accordingly.

Stretch Bound. Let $c > 1$ be a constant (to be determined later). Consider a pair $x, y \in X$ such that $(1+\epsilon)^{i-1} < d_X(x,y) \leq (1+\epsilon)^i$. We will assume by induction that every pair x', y' at distance at most $(1+\epsilon)^{i-1}$ already enjoys stretch at most $\alpha = t \cdot (2 + c \cdot \epsilon)$ in H. Set $\Delta_i = (1+\epsilon)^i$, and let $\tilde{x}, \tilde{y} \in N_i$ be net points such that $d_X(x, \tilde{x}), d_X(y, \tilde{y}) \leq \epsilon \cdot \Delta_i$. By the triangle inequality $d_X(\tilde{x}, \tilde{y}) \leq (1+2\epsilon) \cdot \Delta_i = \Delta$. Therefore there is a $t \cdot \Delta$-bounded partition $\mathcal{P}$ constructed at round i such that $\mathcal{P}(\tilde{x}) = \mathcal{P}(\tilde{y})$. In particular, there is a center vertex $v = v_{\mathcal{P}(\tilde{x})}$ such that both $\{\tilde{x}, v\}, \{\tilde{y}, v\}$ were added to the spanner H. Using the induction hypothesis on the pairs $\{x, \tilde{x}\}$ and $\{y, \tilde{y}\}$, we conclude

$$\begin{aligned}
d_H(x,y) &\leq d_H(x,\tilde{x}) + d_H(\tilde{x}, v) + d_H(v, \tilde{y}) + d_H(\tilde{y}, y) \\
&\leq \alpha \cdot \epsilon\Delta_i + (1+2\epsilon)t\Delta_i + (1+2\epsilon)t\Delta_i + \alpha \cdot \epsilon\Delta_i \\
&\overset{(*)}{<} \frac{\alpha}{1+\epsilon} \cdot \Delta_i \leq \alpha \cdot d_X(x,y) \ ,
\end{aligned}$$

where the inequality $(*)$ follows as $2(1+2\epsilon)t < \alpha(\frac{1}{1+\epsilon} - 2\epsilon)$ for large enough constant c, using that $\epsilon < 1/8$.

Sparsity bound. For a point $x \in X$, let s_x be the maximal index such that $x \in N_{s_x}$. Note that the number of edges in our spanner is not affected by the choice of "cluster centers" in line 5 in Algorithm 1. Therefore, the edge count will be still valid if we assume that $v_P \in P$ is the vertex y with maximal value s_y among all vertices in P.

Consider an edge $\{x, y\}$ added during the i's phase of the algorithm. Necessarily $x, y \in N_i$, and x, y belong to the same cluster P of a partition $\mathcal{P}_j$. W.l.o.g, $y = v_P$, in particular $s_x \le s_y$. The edge $\{x, y\}$ will be charged upon x. Since the partitions at level i are $t \cdot \Delta$ bounded, we have that $d_X(x, y) \le t \cdot \Delta = t \cdot (1 + 2\epsilon) \cdot (1 + \epsilon)^i$. Hence, for i' such that $\epsilon \cdot (1 + \epsilon)^{i'} > t \cdot (1 + 2\epsilon) \cdot (1 + \epsilon)^i$, i.e. $i' > i + O_\epsilon(\log t)$, the points x, y cannot both belong to $N_{i'}$. As $s_x \le s_y$, it must be that $x \notin N_{i'}$. We conclude that x can be charged in at most $O_\epsilon(\log t)$ different levels. As in level i each vertex is charged for at most $\varphi_i \le O(\frac{\log n}{\delta})$ edges, the total charge for each vertex is bounded by $O_\epsilon(\frac{\log n \cdot \log t}{\delta})$.

Lightness bound. Consider the scale $\Delta_i = (1 + \epsilon)^i$. As N_i is an $\epsilon \cdot \Delta_i$-net, Claim 1 implies that N_i has size $n_i \le \frac{2L}{\epsilon \cdot \Delta_i}$, and in any case at most n. In that scale, we constructed $\varphi_i = \frac{2}{\delta} \log n_i \le \frac{2}{\delta} \log n$ partitions, adding at most n_i edges per partition. The weight of each edge added in this scale is bounded by $O(t \cdot \Delta_i)$.

Let H_1 consist of all the edges added in scales $i \in \{\log_{1+\epsilon} \frac{L}{n}, \dots, \log_{1+\epsilon} L\}$, while H_2 consist of edges added in the lower scales. Note that $H = H_1 \cup H_2$.

$$\begin{aligned} w(H_1) &\le \sum_{i \in \left\{\log_{1+\epsilon} \frac{L}{n}, \dots, \log_{1+\epsilon} L\right\}} O(t \cdot \Delta_i) \cdot n_i \cdot \varphi_i \\ &= O\left(\frac{t}{\delta} \cdot \log n \cdot \sum_{i \in \left\{\log_{1+\epsilon} \frac{L}{n}, \dots, \log_{1+\epsilon} L\right\}} \Delta_i \cdot \frac{L}{\epsilon \cdot \Delta_i}\right) = O_\epsilon\left(\frac{t}{\delta} \cdot \log^2 n\right) \cdot L \, . \\ w(H_2) &\le \sum_{\Delta_i \in \frac{L}{n} \cdot \{(1+\epsilon)^{-1}, (1+\epsilon)^{-2}, \dots,\}} O(t \cdot \Delta_i) \cdot n_i \cdot \varphi_i \\ &= O\left(\frac{t}{\delta} \cdot \log n \cdot \sum_{i \ge 1} \frac{1}{(1+\epsilon)^i}\right) \cdot L = O_\epsilon\left(\frac{t}{\delta} \cdot \log n\right) \cdot L \, . \end{aligned}$$

The bound on the lightness follows. ◀

4 Corollaries and Extensions

In this section we describe some corollaries of Theorem 4 for certain metric spaces, and show some extensions, such as improved lightness bound for normed spaces, and discuss graph spanners.

4.1 High Dimensional Normed Spaces

Here we consider the case that the given metric space (X, d) satisfies that every sub-metric $Y \subseteq X$ of size $|Y| = n$ is (t, δ)-decomposable for $\delta = n^{-\beta}$, where $\beta = \beta(t) \in (0, 1)$ is a function of t. In such a case we are able to shave a $\log n$ factor in the lightness.

▶ **Theorem 5.** *Let (X, d_X) be an n-point metric space such that every $Y \subseteq X$ is $(t, |Y|^{-\beta})$-decomposable. Then for every $\epsilon \in (0, 1/8)$, there is a $t \cdot (2 + \epsilon)$-spanner for X with lightness $O_\epsilon\left(\frac{t}{\beta} \cdot n^\beta \cdot \log n\right)$ and sparsity $O_\epsilon\left(n^{1+\beta} \cdot \log n \cdot \log t\right)$.*

Proof. Using the same Algorithm 1, the analysis of the stretch and sparsity from Theorem 4 is still valid, since the number partitions taken in each scale is smaller than in Theorem 4. Recall that in scale i we set $\Delta_i = (1+\epsilon)^i$, and the size of the $\epsilon \cdot \Delta_i$-net N_i is $n_i \le \max\{\frac{2L}{\epsilon \Delta_i}, n\}$. The difference from the previous proof is that N_i is $(t, n_i^{-\beta})$-decomposable, so the number of partitions taken is $\varphi_i = O(n_i^\beta \log n_i)$. In each partition we might add at most one edge per net point, and the weight of this edge is $O(t \cdot \Delta_i)$. We divide the edges of H to H_1 and H_2, and bound the weight of H_2 as above (using that $n_i \le n$). For H_1 we get,

$$\begin{aligned}
w(H_1) &\le \sum_{i \in \left\{\log_{1+\epsilon} \frac{L}{n}, \ldots, \log_{1+\epsilon} L\right\}} O(t \cdot \Delta_i) \cdot n_i \cdot \varphi_i \\
&= O\left(t \cdot \sum_{i \in \left\{\log_{1+\epsilon} \frac{L}{n}, \ldots, \log_{1+\epsilon} L\right\}} \Delta_i \cdot \frac{L}{\epsilon \cdot \Delta_i} \cdot \left(\frac{L}{\epsilon \cdot \Delta_i}\right)^\beta \log \frac{L}{\epsilon \cdot \Delta_i} \right) \\
&= O_\epsilon \left(t \cdot \sum_{i \in \left\{\log_{1+\epsilon} \frac{L}{n}, \ldots, \log_{1+\epsilon} L\right\}} \left(\frac{L}{\Delta_i}\right)^\beta \cdot \log \frac{L}{\Delta_i} \right) \cdot L \\
&= O_\epsilon \left(t \cdot \sum_{i \in \left\{0, \ldots, \log_{1+\epsilon} n\right\}} (i+1) \cdot \left((1+\epsilon)^\beta\right)^i \right) \cdot L \ .
\end{aligned}$$

Set the function $f(x) = \sum_{i=0}^{k} (i+1) \cdot x^i$, on the domain $(1, \infty)$, with parameter $k = \log_{1+\epsilon} n$. Then,

$$\begin{aligned}
f(x) = \left(\int f dx\right)' = \left(\sum_{i=0}^{k} x^{i+1}\right)' &= \left(\frac{x^{k+2} - x}{x-1}\right)' \\
&= \frac{\left((k+2)\, x^{k+1} - 1\right)(x-1) - \left(x^{k+2} - x\right)}{(x-1)^2} \le \frac{(k+2)\, x^{k+1}}{x-1} \ .
\end{aligned}$$

Hence,

$$\begin{aligned}
w(H_1) &= O_\epsilon \left(t \cdot f\left((1+\epsilon)^\beta\right)\right) \cdot L \\
&= O_\epsilon \left(t \cdot \frac{\log_{1+\epsilon} n \cdot \left((1+\epsilon)^\beta\right)^{\log_{1+\epsilon} n}}{(1+\epsilon)^\beta - 1} \right) \cdot L = O_\epsilon \left(\frac{t}{\beta} \cdot n^\beta \cdot \log n \right) \cdot L \ .
\end{aligned}$$

We conclude that the lightness of H is bounded by $O_\epsilon\left(\frac{t}{\beta} \cdot n^\beta \cdot \log n\right)$. ◀

In Section 5 we will show that any n-point Euclidean metric is $(t, n^{-O(1/t^2)})$-decomposable, and that for fixed $p \in (1,2)$, any n-point subset of ℓ_p is $(t, n^{-O(\log^2 t / t^p)})$-decomposable. The following corollaries are implied by Theorem 5 (rescaling t by a constant factor allows us to remove the $O(\cdot)$ term in the exponent of n, while obtaining stretch $O(t)$).

▶ **Corollary 6.** *For a set X of n points in Euclidean space, $t > 1$, there is an $O(t)$-spanner with lightness $O\left(t^3 \cdot n^{1/t^2} \cdot \log n\right)$ and $O\left(n^{1+1/t^2} \cdot \log n \cdot \log t\right)$ edges.*

▶ **Corollary 7.** *For a constant $p \in (1,2)$ and a set X of n points in ℓ_p space, there is an $O(t)$-spanner with lightness $O\left(\frac{t^{1+p}}{\log^2 t} \cdot n^{\log^2 t / t^p} \cdot \log n\right)$ and $O\left(n^{1+\log^2 t / t^p} \cdot \log n \cdot \log t\right)$ edges.*

▶ Remark. Corollary 6 applies for a set of points $X \subseteq \mathbb{R}^d$, where the dimension d is arbitrarily large. If $d = o(\log n)$ we can obtain improved spanners. Specifically, n-point subsets of d-dimensional Euclidean space are $(O(t), 2^{-d/t^2})$-decomposable (see Section 6). Applying Theorem 4 we obtain an $O(t)$-spanner with lightness $O_\epsilon\left(t \cdot 2^{d/t^2} \cdot \log^2 n\right)$ and $O_\epsilon\left(n \cdot 2^{d/t^2} \cdot \log n \cdot \log t\right)$ edges.

4.2 Doubling Metrics

It was shown in [1] that metrics with doubling constant λ are $(t, \lambda^{-O(1/t)})$-decomposable (the case $t = \Theta(\log \lambda)$ was given by [33]). Therefore, Theorem 4 implies:

▶ **Corollary 8.** *For every metric space (X, d_X) with doubling constant λ, and $t \geq 1$, there exist an $O(t)$-spanner with lightness $O\left(t \cdot \log^2 n \cdot \lambda^{1/t}\right)$ and $O\left(n \cdot \lambda^{1/t} \cdot \log n \cdot \log t\right)$ edges.*

4.3 Graph Spanners

In the case where the input is a graph G, it is natural to require that the spanner will be a *graph-spanner*, i.e., a subgraph of G. Given a (metric) spanner H, one can define a graph-spanner H' by replacing every edge $\{x, y\} \in H$ with the shortest path from x to y in G. It is straightforward to verify that the stretch and lightness of H' are no larger than those of H (however, the number of edges may increase).

Consider a graph G with genus g. In [3] it was shown that (the shortest path metric of) G is $\left(t, g^{-O(1/t)}\right)$-decomposable. Furthermore, graphs with genus g have $O(n+g)$ edges [32], so any graph-spanner will have at most so many edges. By Theorem 4 we have:

▶ **Corollary 9.** *Let G be a weighted graph on n vertices with genus g. Given a parameter $t \geq 1$, there exist an $O(t)$-graph-spanner of G with lightness $O\left(t \cdot \log^2 n \cdot g^{1/t}\right)$ and $O(n+g)$ edges.*

For general graphs, the transformation to graph-spanners described above may arbitrarily increase the number of edges (in fact, it will be bounded by $O(\sqrt{|E_H|} \cdot n)$, [20]). Nevertheless, if we have a *strong-decomposition*, we can modify Algorithm 1 to produce a sparse spanner. In a graph $G = (X, E)$, the *strong-diameter* of a cluster $A \subseteq X$ is $\max_{v,u \in A} d_{G[A]}(v, u)$, where $G[A]$ is the induced graph by A (as opposed to weak diameter, which is computed w.r.t the original metric distances). A partition $\mathcal{P}$ of X is Δ-*strongly-bounded* if the strong diameter of every $P \in \mathcal{P}$ is at most Δ. A distribution $\mathcal{D}$ over partitions of X is (t, Δ, δ)-*strong*-decomposition, if it is (t, Δ, δ)-decomposition and in addition every partition $\mathcal{P} \in \text{supp}(\mathcal{D})$ is Δ-strongly-bounded. A graph G is (t, δ)-*strongly-decomposable*, if for every $\Delta > 0$, the graph admits a $(\Delta, t \cdot \Delta, \delta)$-strong-decomposition.

▶ **Theorem 10.** *Let $G = (V, E, w)$ be a (t, δ)-strongly-decomposable, n-vertex graph with aspect ratio $\Lambda = \frac{\max_{e \in E} w(e)}{\min_{e \in E} w(e)}$. Then for every $\epsilon \in (0, 1)$, there is a $t \cdot (2 + \epsilon)$-graph-spanner for G with lightness $O_\epsilon\left(\frac{t}{\delta} \cdot \log^2 n\right)$ and $O_\epsilon(\frac{n}{\delta} \cdot \log n \cdot \log \Lambda)$ edges.*

Proof. We will execute Algorithm 1 with several modifications:
1. The for loop (in Line 2) will go over scales $i \in \{0, \ldots, \log_{1+\epsilon} \Lambda\}$ (instead $\{0, \ldots, \log_{1+\epsilon} L\}$).
2. We will use strong-decompositions instead of regular (weak) decompositions.
3. The partitions created in Line 3 will be over the set of all vertices V, rather then only net points N_i (as otherwise it will be impossible to get strong diameter).
   However, the requirement from close pairs to be clustered together (at least once), is still applied to net points only. Similarly to Claim 3, $\varphi_i = (2 \ln n_i)/\delta$ repetitions will suffice.

4. In Line 6, we will no longer add edges from v_P to all the net points in $P \in \mathcal{P}_j$. Instead, for every net point $x \in P \cap N_i$, we will add a shortest path in $G[P]$ from v_P to x. Note that all the edges added in all the clusters constitute a forest. Thus we add at most n edges per partition.

We now prove the stretch, sparsity and lightness of the resulting spanner.

Stretch. By the triangle inequality, it is enough to show small stretch guarantee only for edges (that is, only for $x, y \in V$ s.t. $\{x, y\} \in E$.) As we assumed that the minimal distance is 1, all the weights are within $[1, \Lambda]$. In particular, every edge $\{x, y\} \in E$ has weight $(1+\epsilon)^{i-1} < w \leq (1+\epsilon)^i$ for $i \in \{0, \dots, \log_{1+\epsilon} \Lambda\}$. The rest of the analysis is similar to Theorem 4, with the only difference being that we use a path from v_P to $\tilde{x}$ rather than the edge $\{\tilde{x}, v_P\}$. This is fine since we only require that the length of this path is at most $(t \cdot (1+2\epsilon) \cdot \Delta)$, which is guaranteed by the strong diameter of clusters.

Sparsity. We have $O_\epsilon(\log \Lambda)$ scales. In each scale we had at most $\varphi_i \leq \frac{2}{\delta} \log n$ partitions, where for each partition we added at most n edges. The bound on the sparsity follows.

Lightness. Consider scale i. We have n_i net points. For each net point we added at most one shortest path of weight at most $O(t \cdot \Delta_i)$ (as each cluster is $O(t \cdot \Delta_i)$-strongly bounded). As the number of partitions is φ_i, the total weight of all edges added at scale i is bounded by $O(t \cdot \Delta_i) \cdot n_i \cdot \varphi_i$. The rest of the analysis follows by similar lines to Theorem 4 (noting that $\Lambda < L$). ◀

5 LSH Induces Decompositions

In this section, we prove that LSH (locality sensitive hashing) induces decompositions. In particular, using the LSH schemes of [5, 46], we will get decompositions for ℓ_2 and ℓ_p spaces, $1 < p < 2$.

▶ **Definition 11.** (Locality-Sensitive-Hashing) Let H be a family of hash functions mapping a metric (X, d_X) to some universe U. We say that H is (r, cr, p_1, p_2)-sensitive if for every pair of points $x, y \in X$, the following properties are satisfied:
1. If $d_X(x, y) \leq r$ then $\Pr_{h \in H}[h(x) = h(y)] \geq p_1$.
2. If $d_X(x, y) > cr$ then $\Pr_{h \in H}[h(x) = h(y)] \leq p_2$.

Given an LSH, its parameter is $\gamma = \frac{\log 1/p_1}{\log 1/p_2}$. We will implicitly always assume that $p_1 \geq n^{-\gamma}$ ($n = |X|$), as indeed will occur in all the discussed settings. Andoni and Indyk [5] showed that for Euclidean space (ℓ_2), and large enough $t > 1$, there is an LSH with parameter $\gamma = O\left(\frac{1}{t^2}\right)$. Nguyen [46], showed that for constant $p \in (1, 2)$, and large enough $t > 1$, there is an LSH for ℓ_p, with parameter $\gamma = O\left(\frac{\log^2 t}{t^p}\right)$. We start with the following claim.

▶ **Claim 12.** *Let (X, d_X) be a metric space, such that for every $r > 0$, there is an $(r, t \cdot r, p_1, p_2)$-sensitive LSH family with parameter γ. Then there is an $\left(r, t \cdot r, n^{-O(\gamma)}, n^{-2}\right)$-sensitive LSH family for X.*

Proof. Set $k = \left\lceil \log_{\frac{1}{p_2}} n^2 \right\rceil \leq \frac{O(\log n)}{\log \frac{1}{p_2}}$, and let H be the promised $(r, t \cdot r, p_1, p_2)$-sensitive LSH family. We define an LSH family H' as follows. In order to sample $h \in H'$, pick $h_1, \dots, h_k$ uniformly and independently at random from H. The hash function h is defined as the

concatenation of $h_1, \dots, h_k$. That is, $h(x) = (h_1(x), \dots, h_k(x))$.
For $x, y \in X$ such that $d_X(x, y) \geq t \cdot r$ it holds that

$$\Pr[h(x) = h(y)] = \Pi_i \Pr[h_i(x) = h_i(y)] \leq p_2^k \leq n^{-2} .$$

On the other hand, for $x, y \in X$ such that $d_X(x, y) \leq r$, it holds that

$$\Pr[h(x) = h(y)] = \Pi_i \Pr[h_i(x) = h_i(y)] \geq p_1^k = 2^{-\log\frac{1}{p_1}\cdot\frac{O(\log n)}{\log\frac{1}{p_2}}} = n^{-O(\gamma)} . \quad ◀$$

▶ **Lemma 13.** *Let (X, d_X) be a metric space, such that for every $r > 0$, there is a $(r, t \cdot r, p_1, p_2)$-sensitive LSH family with parameter γ. Then (X, d_X) is $(t, n^{-O(\gamma)})$-decomposable.*

Proof. Let H' be an $\left(r, tr, n^{-O(\gamma)}, n^{-2}\right)$-sensitive LSH family, given by Claim 12. We will use H' in order to construct a decomposition for X. Each hash function $h \in H'$ induces a partition $\mathcal{P}_h$, by clustering all points with the same hash value, i.e. $\mathcal{P}_h(x) = \mathcal{P}_h(y) \iff h(x) = h(y)$. However, in order to ensure that our partition will be $t \cdot r$-bounded, we modify it slightly. For $x \in X$, if there is a $y \in \mathcal{P}_h(x)$ with $d_X(x, y) > t \cdot r$, remove x from $\mathcal{P}_h(x)$, and create a new cluster $\{x\}$. Denote by $\mathcal{P}'_h$ the resulting partition. $\mathcal{P}'_h$ is clearly $t \cdot r$-bounded, and we argue that every pair x, y at distance at most r is clustered together with probability at least $n^{-O(\gamma)}$. Denote by χ_x (resp., χ_y) the probability that x (resp., y) was removed from $\mathcal{P}_h(x)$ (resp., $\mathcal{P}_h(y)$). By the union bound on the at most n points in $\mathcal{P}_h(x)$, we have that both $\chi_x, \chi_y \leq 1/n$. We conclude

$$\Pr_{\mathcal{P}'_h}[\mathcal{P}'_h(x) = \mathcal{P}'_h(y)] \geq \Pr_{h\sim H}[h(x) = h(y)] - \Pr_h[\chi_x \vee \chi_y] \geq n^{-O(\gamma)} - \frac{2}{n} = n^{-O(\gamma)} . \quad ◀$$

Using [5], Lemma 13 implies that ℓ_2 is $(t, n^{-O(1/t^2)})$-decomposable. Moreover, using [46] for constant $p \in (1, 2)$, Lemma 13 implies that ℓ_p is $(t, n^{-O(\log^2 t/t^p)})$-decomposable.

6 Decomposition for d-Dimensional Euclidean Space

In Section 5, using a reduction from LSH, we showed that ℓ_2 is $(t, n^{-O(1/t^2)})$-decomposable. Here, we will show that for dimension $d = o(\log n)$, using a direct approach, better decomposition could be constructed.

Denote by $B_d(x, r)$ the d dimensional ball of radius r around x (w.r.t ℓ_2 norm). $V_d(r)$ denotes the volume of $B_d(x, r)$ (note that the center here is irrelevant). Denote by $C_d(u, r)$ the volume of the intersection of two balls of radius r, the centers of which are at distance u (i.e. for $\|x - y\|_2 = u$, $C_d(u, r)$ denotes the volume of $B_d(x, r) \cap B_d(y, r)$). We will use the following lemma which was proved in [5] (based on a lemma from [27]).

▶ **Lemma 14.** *([5]) For any $d \geq 2$ and $0 \leq u \leq r$*

$$\Omega\left(\frac{1}{\sqrt{d}}\right) \cdot \left(1 - \left(\frac{u}{r}\right)^2\right)^{\frac{d}{2}} \leq \frac{C_d(u, r)}{V_d(r)} \leq \left(1 - \left(\frac{u}{r}\right)^2\right)^{\frac{d}{2}} .$$

Using Lemma 14, we can construct better decompositions:

▶ **Lemma 15.** *For every $d \geq 2$ and $2 \leq t \leq \sqrt{2d/\ln d}$, ℓ_2^d is $O(t, 2^{-O(\frac{d}{t^2})})$-decomposable.*

Proof. Consider a set X of n points in ℓ_2^d, and fix $r > 0$. Let $\mathcal{B}$ be some box which includes all of X and such that each $x \in X$ is at distance at least $t \cdot r$ from the boundary of B. We sample points $s_1, s_2 \dots$ uniformly at random from $\mathcal{B}$. Set $P_i = B_X(s_i, \frac{t \cdot r}{2}) \setminus \bigcup_{j=1}^{i-1} B_X\left(s_j, \frac{t \cdot r}{2}\right)$. We sample points until $X = \bigcup_{i\geq 1} P_i$. Then, the partition will be $\mathcal{P} = \{P_1, P_2, \ldots.\}$ (dropping empty clusters).

It is straightforward that $\mathcal{P}$ is $t\cdot r$-bounded. Thus it will be enough to prove that every pair x, y at distance at most r, has high enough probability to be clustered together. Let s_i be the first point sampled in $B_d\left(x, \frac{t\cdot r}{2}\right) \cup B_d\left(y, \frac{t\cdot r}{2}\right)$. By the minimality of i, $x, y \notin \bigcup_{j=1}^{i-1} B_d\left(s_j, \frac{t\cdot r}{2}\right)$ and thus both are yet un-clustered. If $s_i \in B_d\left(x, \frac{t\cdot r}{2}\right) \cap B_d\left(y, \frac{t\cdot r}{2}\right)$ then both x, y join P_i and thus clustered together. Using Lemma 14 we conclude,

$$\begin{aligned}
\Pr_{\mathcal{P}}\left[\mathcal{P}(x) = \mathcal{P}(y)\right] &= \Pr\left[s_i \in B_d\left(x, \frac{t\cdot r}{2}\right) \cap B_d\left(y, \frac{t\cdot r}{2}\right)\right.\\
&\qquad\qquad \left.\middle| s_i \text{ is first in } B_d\left(x, \frac{t\cdot r}{2}\right) \cup B_d\left(y, \frac{t\cdot r}{2}\right)\right]\\
&\geq \frac{C_d(\|x-y\|_2, \frac{t\cdot r}{2})}{2\cdot V_d(\frac{t\cdot r}{2})}\\
&= \Omega\left(\frac{1}{\sqrt{d}}\right)\left(1 - \left(\frac{\|x-y\|_2}{\frac{t\cdot r}{2}}\right)^2\right)^{\frac{d}{2}}\\
&= \Omega\left(\frac{1}{\sqrt{d}}\right)\left(1 - \frac{4}{t^2}\right)^{\frac{d}{2}}\\
&= \Omega\left(e^{-\frac{2d}{t^2} - \frac{1}{2}\ln d}\right) = 2^{-O(d/t^2)}\ .
\end{aligned}$$ ◀

References

1 Ittai Abraham, Yair Bartal, and Ofer Neiman. Advances in metric embedding theory. *Advances in Mathematics*, 228(6):3026–3126, 2011. doi:10.1016/j.aim.2011.08.003.

2 Ittai Abraham, Shiri Chechik, Michael Elkin, Arnold Filtser, and Ofer Neiman. Ramsey spanning trees and their applications. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, Louisiana, USA, January 7-10*, 2018.

3 Ittai Abraham, Cyril Gavoille, Anupam Gupta, Ofer Neiman, and Kunal Talwar. Cops, robbers, and threatening skeletons: padded decomposition for minor-free graphs. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 79–88, 2014. doi:10.1145/2591796.2591849.

4 Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993. doi:10.1007/BF02189308.

5 Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 459–468, 2006. doi:10.1109/FOCS.2006.49.

6 Baruch Awerbuch. Communication-time trade-offs in network synchronization. In *Proc. of 4th PODC*, pages 272–276, 1985.

7 Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, 1985. doi:10.1145/4221.4227.

8 Baruch Awerbuch, Alan E. Baratz, and David Peleg. Cost-sensitive analysis of communication protocols. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing, Quebec City, Quebec, Canada, August 22-24, 1990*, pages 177–187, 1990. doi:10.1145/93385.93417.

9 Baruch Awerbuch, Alan E. Baratz, and David Peleg. Efficient broadcast and light-weight spanners. *Manuscript*, 1991.

10 Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *Proc. of 37th FOCS*, pages 184–193, 1996.

11 Punyashloka Biswal, James R. Lee, and Satish Rao. Eigenvalue bounds, spectral partitioning, and metrical deformations via flows. *J. ACM*, 57(3), 2010. `doi:10.1145/1706591.1706593`.

12 Glencora Borradaile, Hung Le, and Christian Wulff-Nilsen. Greedy spanners are optimal in doubling metrics. *CoRR*, abs/1712.05007, 2017. `arXiv:1712.05007`.

13 Glencora Borradaile, Hung Le, and Christian Wulff-Nilsen. Minor-free graphs have light spanners. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 767–778, 2017. `doi:10.1109/FOCS.2017.76`.

14 R. Braynard, D. Kostic, A. Rodriguez, J. Chase, and A. Vahdat. Opus: an overlay peer utility service. In *Prof. of 5th OPENARCH*, 2002.

15 Gruia Calinescu, Howard Karloff, and Yuval Rabani. Approximation algorithms for the 0-extension problem. *SIAM J. Comput.*, 34(2):358–372, 2005. `doi:10.1137/S0097539701395978`.

16 P. B. Callahan and S. R. Kosaraju. A decomposition of multi-dimensional point-sets with applications to k-nearest-neighbors and n-body potential fields. In *Proc. of 24th STOC*, pages 546–556, 1992.

17 B. Chandra, G. Das, G. Narasimhan, and J. Soares. New sparseness results on graph spanners. *Int. J. Comput. Geometry Appl.*, 5:125–144, 1995.

18 Shiri Chechik and Christian Wulff-Nilsen. Near-optimal light spanners. In *Proc. of 27th SODA*, pages 883–892, 2016.

19 Edith Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM J. Comput.*, 28(1):210–236, 1998. `doi:10.1137/S0097539794261295`.

20 Don Coppersmith and Michael Elkin. Sparse sourcewise and pairwise distance preservers. *SIAM J. Discrete Math.*, 20(2):463–501, 2006. `doi:10.1137/050630696`.

21 Gautam Das, Paul J. Heffernan, and Giri Narasimhan. Optimally sparse spanners in 3-dimensional euclidean space. In *Proceedings of the Ninth Annual Symposium on Computational GeometrySan Diego, CA, USA, May 19-21, 1993*, pages 53–62, 1993. `doi:10.1145/160985.160998`.

22 Amin Vahdat Dejan Kostic. Latency versus cost optimizations in hierarchical overlay networks. Technical Report CS-2001-04, Duke University, 2002.

23 Michael Elkin. Computing almost shortest paths. *ACM Trans. Algorithms*, 1(2):283–323, 2005. `doi:10.1145/1103963.1103968`.

24 Michael Elkin, Ofer Neiman, and Shay Solomon. Light spanners. In *Proc. of 41th ICALP*, pages 442–452, 2014.

25 Michael Elkin and Jian Zhang. Efficient algorithms for constructing (1+epsilon, beta)-spanners in the distributed and streaming models. *Distributed Computing*, 18(5):375–385, 2006. `doi:10.1007/s00446-005-0147-2`.

26 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, STOC '03, pages 448–455, New York, NY, USA, 2003. ACM. `doi:10.1145/780542.780608`.

27 Uriel Feige and Gideon Schechtman. On the optimality of the random hyperplane rounding technique for max cut. *Random Struct. Algorithms*, 20(3):403–440, 2002. `doi:10.1002/rsa.10036`.

28 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the streaming model: the value of space. In *Proc. of 16th SODA*, pages 745–754, 2005.

29 Arnold Filtser and Shay Solomon. The greedy spanner is existentially optimal. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 9–17, 2016. doi:10.1145/2933057.2933114.

30 Lee-Ad Gottlieb. A light metric spanner. In *Proc. of 56th FOCS*, pages 759–772, 2015.

31 Michelangelo Grigni. Approximate TSP in graphs with forbidden minors. In *Proc. of 27th ICALP*, pages 869–877, 2000.

32 Jonathan L. Gross and Thomas W. Tucker. *Topological Graph Theory*. Wiley-Interscience, New York, NY, USA, 1987.

33 Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *Proc. of 44th FOCS*, pages 534–543, 2003.

34 Sariel Har-Peled, Piotr Indyk, and Anastasios Sidiropoulos. Euclidean spanners in high dimensions. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 804–809, 2013. doi:10.1137/1.9781611973105.57.

35 Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006. doi:10.1137/S0097539704446281.

36 Jonathan A. Kelner, James R. Lee, Gregory N. Price, and Shang-Hua Teng. Higher eigenvalues of graphs. In *FOCS*, pages 735–744, 2009. doi:10.1109/FOCS.2009.69.

37 Philip N. Klein, Serge A. Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *STOC*, pages 682–690, 1993. doi:10.1145/167088.167261.

38 Robert Krauthgamer, James R. Lee, Manor Mendel, and Assaf Naor. Measured descent: A new embedding method for finite metrics. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 434–443, Washington, DC, USA, 2004. IEEE Computer Society. doi:10.1109/FOCS.2004.41.

39 J. R. Lee and A. Naor. Extending lipschitz functions via random metric partitions. *Inventiones Mathematicae*, 160(1):59–95, 2005.

40 James R. Lee and Anastasios Sidiropoulos. Genus and the geometry of the cut graph. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 193–201, 2010. doi:10.1137/1.9781611973075.18.

41 Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46:787–832, November 1999. doi:10.1145/331524.331526.

42 N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

43 Nathan Linial and Michael Saks. Low diameter graph decompositions. *Combinatorica*, 13(4):441–454, 1993. (Preliminary version in *2nd SODA*, 1991).

44 Manor Mendel and Assaf Naor. Ramsey partitions and proximity data structures. *Journal of the European Mathematical Society*, 9(2):253–275, 2007.

45 Giri Narasimhan and Michiel H. M. Smid. *Geometric spanner networks*. Cambridge University Press, 2007.

46 Huy L. Nguyen. Approximate nearest neighbor search in ℓ_p. *CoRR*, abs/1306.3601, 2013. arXiv:1306.3601.

47 David Peleg. Proximity-preserving labeling schemes and their applications. In *Graph-Theoretic Concepts in Computer Science, 25th International Workshop, WG '99, As-*

cona, Switzerland, June 17-19, 1999, Proceedings, pages 30–41, 1999. `doi:10.1007/3-540-46784-X_5`.

48 David Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM, Philadelphia, PA, 2000.

49 David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18(4):740–747, 1989. `doi:10.1137/0218050`.

50 David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989.

51 Satish B. Rao. Small distortion and volume preserving embeddings for planar and Euclidean metrics. In *SOCG*, pages 300–306, 1999.

52 L. Roditty and U. Zwick. On dynamic shortest paths problems. In *Proc. of 32nd ESA*, pages 580–591, 2004.

53 Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, pages 261–272, 2005. `doi:10.1007/11523468_22`.

54 J. S. Salowe. Construction of multidimensional spanner graphs, with applications to minimum spanning trees. In *Proc. of 7th SoCG*, pages 256–261, 1991.

55 Michiel H. M. Smid. The weak gap property in metric spaces of bounded doubling dimension. In *Efficient Algorithms, Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, pages 275–289, 2009. `doi:10.1007/978-3-642-03456-5_19`.

56 Kunal Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 281–290, 2004. `doi:10.1145/1007352.1007399`.

57 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proc. of 13th SPAA*, pages 1–10, 2001.

58 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. `doi:10.1145/1044731.1044732`.

59 P. M. Vaidya. A sparse graph almost as good as the complete graph on points in k dimensions. *Discrete & Computational Geometry*, 6:369–381, 1991.

60 Jürgen Vogel, Jörg Widmer, Dirk Farin, Martin Mauve, and Wolfgang Effelsberg. Priority-based distribution trees for application-level multicast. In *Proceedings of the 2nd Workshop on Network and System Support for Games, NETGAMES 2003, Redwood City, California, USA, May 22-23, 2003*, pages 148–157, 2003. `doi:10.1145/963900.963914`.

61 Bang Ye Wu, Kun-Mao Chao, and Chuan Yi Tang. Light graphs with small routing cost. *Networks*, 39(3):130–138, 2002. `doi:10.1002/net.10019`.

On the Tractability of Optimization Problems on H-Graphs

Fedor V. Fomin[1]
Department of Informatics, University of Bergen, Norway
fedor.fomin@ii.uib.no
https://orcid.org/0000-0003-1955-4612

Petr A. Golovach[2]
Department of Informatics, University of Bergen, Norway
petr.golovach@ii.uib.no
https://orcid.org/0000-0002-2619-2990

Jean-Florent Raymond[3]
Technische Universität Berlin, Germany
raymond@tu-berlin.de
https://orcid.org/0000-0003-4646-7602

Abstract

For a graph H, a graph G is an H-graph if it is an intersection graph of connected subgraphs of some subdivision of H. These graphs naturally generalize several important graph classes like interval graphs or circular-arc graph. This notion was introduced in the early 1990s by Bíró, Hujter, and Tuza. Recently, Chaplick et al. initiated the algorithmic study of H-graphs by showing that a number of fundamental optimization problems like Clique, Independent Set, or Dominating Set are solvable in polynomial time on H-graphs. We extend and complement these algorithmic findings in several directions.

First we show that for every fixed H, the class of H-graphs is of logarithmically-bounded boolean-width. We also prove that H-graphs are graphs with polynomially many minimal separators. Pipelined with the plethora of known algorithms on graphs of bounded boolean-width and graphs with polynomially many minimal separators, this describes a large class of optimization problems that are solvable in polynomial time on H-graphs.

The most fundamental optimization problems among those solvable in polynomial time on H-graphs are Clique, Independent Set, and Dominating Set. We provide a more refined complexity analysis of these problems from the perspective of parameterized complexity. We show that Independent Set and Dominating Set are W[1]-hard being parameterized by the size of H plus the size of the solution. On the other hand, we prove that when H is a tree, Dominating Set is fixed-parameter tractable (FPT) parameterized by the size of H. Besides, we show that Clique admits a polynomial kernel parameterized by H and the solution size.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Theory of computation → Graph algorithms analysis

Keywords and phrases H-topological intersection graphs, parameterized complexity, minimal separators, boolean-width, mim-width

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.30

[1] Supported by the Research Council of Norway via the project "MULTIVAL".
[2] Supported by the Research Council of Norway via the project "CLASSIS".
[3] Supported by the European Research Council (ERC) via the ERC consolidator grant DISTRUCT-648527. This research was partly done while the author was supported by the Polish National Science Centre grant PRELUDIUM DEC-2013/11/N/ST6/02706.

© Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond;
licensed under Creative Commons License CC-BY

Related Version A full version of the paper is available at [13], `https://arxiv.org/abs/1709.09737`.

Acknowledgements We thank Saket Saurabh for bringing H-graphs to our attention.

1 Introduction

The notion of H-graph was introduced in the work of Bíró, Hujter, and Tuza [4] on precoloring extensions of graphs. H-graphs nicely generalize several popular and widely studied classes of graphs. For example, the classical definition of an interval graph is as a graph which is an intersection graph[4] of intervals of a line. Equivalently, a graph is interval if it is an intersection graph of some subpaths of a path. Or, equivalently, if it is an intersection graph of some subgraphs of some subdivision (that is a graph obtained by placing vertices of degree 2 on the edges) of P_2, the graph with two adjacent vertices. More generally, for a fixed graph H, an H-graph is an intersection graph of some connected subgraphs of some subdivision of H. Thus for example, an interval graph is a P_2-graph, a circular-arc graph is a C_2-graph, where C_2 is a double-edge with two endpoints, a split graph is a $K_{1,d}$-graph for some $d \geq 0$, where $K_{1,d}$ is a star with d leaves, a chordal graph is a T-graph for some tree T, etc..

The main motivation behind the study of H-graphs is the following. It is well-known that on interval, chordal, circular-arc, and other graphs with "simple" intersection models many NP-hard optimization problems are solvable in polynomial time, see e.g. the book of Golumbic [17] for an overview. It is a natural question whether at least some of these algorithmic results can be extended to more general classes of intersection graphs. Chaplick et al. [8, 9] initiated the systematic study of algorithmic properties of H-graphs. They showed that a number of fundamental optimization problems like Independent Set and Dominating Set are solvable in polynomial time on H-graphs for any fixed H. Most of their algorithms run on H-graphs in time $n^{f(H)}$, where n is the number of vertices in the input graph and f is some function. In other words, being parameterized by H most of the problems are known to be in the class XP. Our work is driven by the following question.

- Are there generic explanations why many problems admit polynomial time algorithms on H-graphs?

We address the first question by proving the following combinatorial results. We show first that every n-vertex H-graph has boolean-width at most $2|E(H)| \cdot \log n$, and that a decomposition of this width can be found in polynomial time. This combinatorial result extends the results of Belmonte and Vatshelle [2, 1] on the boolean-width of interval and circular-arc graphs to H-graphs. Together with the algorithms for a vast class of problems called *LC-VSP* problems [6, 2], and for problems related to induced paths [21], this implies immediately that all these problems are solvable in polynomial time on H-graphs. The illustrative problems where this approach is successful are Weighted Independent Set, Weighted Dominating Set, Total Dominating Set, Induced Matching, Longest Induced Path and Disjoint Induced Paths, and many others.

Then we prove that every n-vertex H-graph has at most $(2n+1)^{|E(H)|} + |E(H)| \cdot (2n)^2$ minimal separators.[5] Pipelining the bound on the number of minimal separators in H-graphs with meta-algorithmic results of Fomin, Todinca and Villanger [14], we obtained another wide class of problems solvable in polynomial time on H-graphs. Examples of such problems

[4] The intersection graph of a family $\mathcal{S}$ of sets has vertex set $\mathcal{S}$ and edge set $\{SS',\ S \cap S' \neq \emptyset\}$.
[5] It was reported to us by Steven Chaplick and Peter Zeman that they also obtained this result independently and that it will be included in the journal version of their paper.

are Treewidth, Feedback Vertex Set, Maximum Induced Subgraph excluding a planar minor, and various packing problems.

All these generic algorithmic results provide XP algorithms when parameterized by the size of H. This brings us to the second question defining the direction of our research.

- What is the parameterized complexity of the fundamental optimization problems being parameterized by the size of H?

The first steps in this direction were done by Chaplick et al. in [8] who showed that Dominating Set is fixed-parameter tractable (FPT) on $K_{1,d}$-graphs parameterized by d. In this paper we show that Dominating Set is W[1]-hard parameterized by the size of H plus the solution size. Thus the existence of an FPT algorithm for a general graph H is very unlikely. (We refer to books [11, 10] for definitions from parameterized complexity and algorithms.) We also prove a similar lower bound for Independent Set parameterized by the size of H plus the solution size. Combined with our combinatorial results, these lower-bounds show that Independent Set and Dominating Set are also W[1]-hard when parameterized by mim-width (a graph parameter to be defined in the corresponding section) of the input and the solution size. The technique we develop to establish lower bounds on H-graphs found applications beyond the topic of this paper [20, 21].

On the positive side, we show that when H is a tree, then Dominating Set is FPT parameterized by the size of H. This significantly extends the result from [8] for stars to arbitrary trees. We actually prove a slightly more general result, namely that Dominating Set is FPT on chordal graphs G parameterized by the leafage of the graph, i.e. the minimum number of leaves in the clique tree of G.

Finally we show that Clique admits a polynomial kernel when parameterized by the size of H plus the solution size. This strengthens the result of Chaplick et al. [8] who showed that Clique is FPT for such a parameterization.

Organization of the paper. We give hereafter the necessary definitions. In Section 2, we upper-bound the boolean-width of H-graphs and provide algorithmic applications. We address minimal separators of H-graphs in Section 3, again with algorithmic consequences. Last, Section 4 contains our results on the parameterized complexity of some classic optimization problems on H-graphs. Due to space constraints, the proofs of the statements marked with (★) are omitted and some other proofs are just sketched. The full details can be found in the complete version of the paper [13].

Definitions. All graphs in this paper are finite, undirected, loopless, and may have multiple edges. If G is a graph, we denote by $|G|$ and $\|G\|$, respectively, its numbers of vertices and edges (counting multiplicities). If $X, Y \subseteq V(G)$, $\overline{X}$ is the complement of X in $V(G)$ (i.e. $\overline{X} = V(G) \setminus X$), $G[X]$ is the subgraph of G induced by the vertices of X, and $G[X, Y]$ is the bipartite subgraph of G induced by those edges that have one endpoint in X and the other in Y. Unless otherwise stated, logarithms are binary.

In this paper H is always a fixed (multi)graph. We say that a graph G is an H-graph if there is a subdivision H' of H and a collection $\mathcal{M} = \{M_v\}_{v \in V(G)}$ (called an *H-representation* or, simply, *representation*) of subsets of $V(H')$, each inducing a connected subgraph, such that G is isomorphic to the intersection graph of $\mathcal{M}$. To avoid confusion, we refer to the vertices of H and H' as *nodes*. We also say that the nodes of H are *branching* nodes of H' and the other nodes are *subdivision* nodes. If v is a vertex of G, then M_v is the *model* of v in the representation $\mathcal{M}$. For every set $A \subseteq V(G)$, we define $M_A = \bigcup_{v \in A} M_v$. For every node u of H', we denote by V_u the set of vertices of G whose model contains u, i.e. $V_u = \{v \in V(G),\ u \in M_v\}$.

2 H-graphs have logarithmic boolean-width

Boolean-width is a graph invariant that has been introduced in [6] and which is related to the number of different neighborhoods along a cut. Belmonte and Vatshelle showed in [2] that n-vertex interval graphs and circular-arc graphs have boolean-width $\mathcal{O}(\log n)$. In this section, we generalize their result by proving that, for any fixed graph H, n-vertex H-graphs have boolean-width $\mathcal{O}(\log n)$. Using the results of [7, 21], we obtain polynomial algorithms for a vast class of optimization problems on H-graphs. Before we proceed with the proofs, we need to introduce some notions specific to this section.

An *induced matching* in a graph G is a set of vertices that induces a disjoint union of edges. If $X \subseteq V(G)$, $\mathrm{mim}(X)$ denotes the maximum number of edges of an induced matching in $G[X, \overline{X}]$.

Let $d \in \mathbb{N}$ and and let $A \subseteq V(G)$. Two subsets $X, Y \subseteq A$ are said to be *d-neighborhood equivalent*, what we denote by $X \equiv_A^d Y$, if $\min(d, |X \cap N(v)|) = \min(d, |Y \cap N(v)|)$ holds for every $v \in \overline{A}$. We write $\mathrm{nec}_d(A)$ for the number of equivalence classes of the relation $\equiv_A^d$.

A *carving decomposition* of a graph G is a pair (T, δ) where T is a full binary rooted tree (that is, every non-leaf vertex has degree 3) and δ is a bijection from the leaves of T to the vertices of G. A carving decomposition (T, δ) is a *caterpillar decomposition* if T can be obtained from a path by adding a vertex of degree one adjacent to every internal vertex. If $w \in V(T)$, we define V_w as the set of vertices of G in bijection with the leaves of the subtree of T rooted at w. We also denote by $\mathrm{mim}(T, \delta)$ (resp. $\mathrm{nec}_d(T, \delta)$, and $\mathrm{boolw}(T, \delta)$) the maximum of $\mathrm{mim}(V_w)$ (resp. $\max\{\mathrm{nec}_d(V_w), \mathrm{nec}_d(\overline{V_w})\}$, and $\log(\mathrm{nec}_1(V_w))$) taken over all $w \in V(T)$.

The *boolean-width* of (T, δ) is the value $\mathrm{boolw}(T, \delta)$ and the boolean-width of G, denoted by $\mathrm{boolw}(G)$, is the minimum boolean-width of a carving decomposition of G.

The following lemma relates maximum induced matchings to neighborhood equivalence.

▶ **Lemma 1** ([2, Lemma 1 and Lemma 2]). *For every n-vertex graph G and $A \subseteq V(G)$,*
1. *$\mathrm{mim}(A) \leq k$ iff for every $S \subseteq A$ there is a $R \subseteq S$ s.t. $R \equiv_A^1 S$ and $|R| \leq k$;*
2. *$\mathrm{nec}_d(A) \leq n^{d \cdot \mathrm{mim}(A)}$.*

Our results on the boolean-width of H-graphs follow from the next result.

▶ **Theorem 2.** *For every n-vertex H-graph G with $n \geq 2$ whose intersection model is given, we can compute in polynomial time a caterpillar decomposition (T, δ) with $\mathrm{mim}(T, \delta) \leq 2\|H\|$.*

Proof. Let F be the subdivision of H in which G can be realized and let $\{M_v\}_{v \in V(G)}$ be the intersection model of G. Let us arbitrarily fix a branching node r of F. Let $v_1, \ldots, v_n$ be an ordering of $V(G)$ by non-decreasing distance of M_{v_i}'s to r.

▶ **Claim 3.** *For every prefix A of $v_1, \ldots, v_n$ and every $S \subseteq A$, there is a set $R \subseteq S$ of size at most $2\|H\|$ such that $R \equiv_A^1 S$.*

Proof. Let A be a prefix of $v_1, \ldots, v_n$ and let $S \subseteq A$. Let $M_A = \bigcup_{v \in A} M_v$ and similarly for $M_{\overline{A}}$ and M_S. Let us consider the path P_e corresponding to some edge $e \in E(H)$. Let $x_1, \ldots, x_p$ be the vertices of P_e in the same order.

Let $v \in A$ and notice that since, by definition, $G[M_v]$ is connected, the vertex set $M_v \cap V(P_e)$ induces at most two connected components in P_e. Indeed if $M_v \cap V(P_e)$ induced more than two connected components, then one of them would not contain any endpoint of P_e, and thus this component would not be connected to other vertices of M_v in $G[M_v]$. Let us assume that it induces at least one connected component and let x_i and x_j be the first and last vertices (wrt. the ordering $x_1, \dots, x_p$) of this component. If $\{x_1, \dots, x_{i-1}\}$ is disjoint from $M_{\overline{A}}$, we say that v is a *left-protector* of P_e. If j is maximum among all vertices that protects the left of P_e, then v is a *rightmost left-protector*. (Informally, it extends the most to the right.) Similarly, v is a *right-protector* the right of P_e if $\{x_{j+1}, \dots, x_p\}$ is disjoint from $M_{\overline{A}}$ and is a *leftmost right-protector* if i is minimal.

Let Z_e be a set containing one (arbitrarily chosen) rightmost left-protector and one leftmost right-protector of e if some exist, and let $R = \bigcup_{e \in E(H)} Z_e$. Clearly $|R| \leq 2\|H\|$. Let us now show that $N(S) \cap \overline{A} \subseteq N(R) \cap \overline{A}$. We consider a vertex $u \in N(S) \cap \overline{A}$ and we show that it also belongs to $N(R)$. Let v be a neighbor of u in S. As u and v are adjacent, M_u and M_v have non-empty intersection. Let e be an edge of H such that M_u and M_v meet on P_e, i.e. $M_u \cap M_v \cap V(P_e) \neq \emptyset$. Again, we denote by $x_1, \dots, x_p$ the vertices of P_e.

▶ **Claim 4** (★). *Let $w \in A$. If $M_w \cap V(P_e) = \{x_i, \dots, x_j\}$ with $1 \leq i \leq j \leq p$, then one of $\{x_k,\ 1 \leq k < i\}$ and $\{x_k,\ j < k \leq p\}$ is disjoint from $M_{\overline{A}}$.*

As M_u intersects M_v on P_e, it intersects the vertex set C of one component induced by M_v on P_e (recall that there are either one or two such components). In the case where there are two components, we assume without loss of generality that this is the "left" one (i.e. that with smallest indices). In the case where there is one component, we assume that v is a left-protector of P_e (according to Claim 4, v is a left-protector or a right-protector of P_e). Observe that in both cases, v is a left-protector of P_e. Let z be the rightmost left-protector of P_e that belongs to R and let $x_k, \dots x_{k'}$ be the vertices of the corresponding component of $P_e[M_z \cap V(P_e)]$ (that is, the component used in the definition of left-protector).

Notice that $C \subseteq \{x_1, \dots, x_{k'}\}$, by maximality of z (informally, because it is "rightmost"). As z is a left-protector, $M_u \cap \{x_1, \dots, x_{k-1}\} = \emptyset$. Since M_u and C intersect, they intersect in $\{x_k, \dots, x_{k'}\}$. Therefore $M_u \cap M_z \neq \emptyset$: z is adjacent to u. As $z \in R$, we are done. ◀

We construct a caterpillar decomposition that follows the ordering $v_1, \dots, v_n$. If $n = 2$, then we define T to be the tree with the two vertices and δ maps them to v_1 and v_2. Assume that $n \geq 3$. We construct a path $x_2 \dots x_{n-1}$ and n vertices $y_1, \dots, y_n$. Then we make y_1, y_2 adjacent to x_2, y_i is made adjacent to x_i if $3 \leq i \leq n-2$, and y_{n_1}, y_n is adjacent to x_{n-1}. We define $\delta(y_i) = v_i$ for $i \in \{1, \dots, n\}$. In both cases the root is chosen arbitrarily. According to Claim 3 and Lemma 1.(1), this caterpillar decomposition satisfies $\mathrm{mim}(T, \delta) \leq 2\|H\|$. ◀

The next result follows from the application to the decomposition provided by Theorem 2 of Lemma 1.(2), with the fact that $\mathrm{mim}(A) = \mathrm{mim}(\overline{A})$ for every $A \subseteq V(G)$.

▶ **Corollary 5.** *For every n-vertex H-graph G with $n \geq 2$ whose intersection model is given, we can compute in polynomial time a caterpillar decomposition (T, δ) with $\mathrm{nec}_d(T, \delta) \leq n^{d \cdot 2\|H\|}$.*

From the definition of boolean-width, we also get:

▶ **Corollary 6.** *Every n-vertex H-graph with $n \geq 2$ has boolean-width at most $2\|H\| \cdot \log n$.*

By choosing H to be a single or double edge, we recover the results of [2] on the boolean-width of interval and circular-arc graphs, respectively, as special cases of Corollary 6. Apart of the degenerate case where H is edgeless (in which case H-graphs are disjoint unions of cliques), every interval graph is an H-graph. Hence the $\Omega(\log n)$ lower bound of [2] shows that Corollary 6 is tight up to a constant factor.

Algorithmic applications. Boolean-width and nec_d have been used in [6, 7] to design parameterized algorithms for the problems Weighted Independent Set, Weighted Dominating Set, and a vast class of problems, called *LC-VSP* problems, that includes fundamental problems as Independent Set, Independent Dominating Set, Total Dominating Set, Induced Matching, and many others (see [7]). The main result of [7] is the following.

▶ **Theorem 7** ([7]). *For every LC-VSP problem Π, there are constants d and q such that Π can be solved in time $\mathcal{O}(n^4 \cdot q \cdot \mathrm{nec}_d(T,\delta)^{3q})$ if a decomposition (T,δ) of the input is given.*

Recently, Jaffke, Kwon, and Telle obtained polynomial-time algorithms on graphs of bounded mim-width for problems that are not LC-VSP.

▶ **Theorem 8** ([21]). *The problems* Longest Induced Path, Induced Disjoint Paths, *and H-*Induced Subdivision[6] *can be solved in time $n^{\mathcal{O}(\mathrm{mim}(T,\delta))}$ if a decomposition (T,δ) of the input is given.*

Combining Theorem 2 with Theorem 8 and Corollary 5 with Theorem 7, we get the following meta-algorithmic consequences.

▶ **Theorem 9.** *Let H be a graph and let Π be either a LC-VSP problem or one of* Longest Induced Path, Induced Disjoint Paths, *and H-*Induced Subdivision. *Then Π can be solved in polynomial time on H-graphs if an H-representation of the input is provided.*

3 H-graphs have few minimal separators

Let G be a graph. A set $X \subseteq V(G)$ is a *minimal separator* of G if $G \setminus X$ has more connected components than G, and X is inclusion-minimal with this property. The study of minimal separators is an active line of research that found many algorithmic applications (see e.g. [22, 3, 5, 14]). In general, the number of minimal separators of a graph may be as large as exponential in its number of vertices. We prove in this section that in an H-graph, this number is upper-bounded by a polynomial (Theorem 10). Combining this finding with the meta-algorithmic results of [14], we deduce that a wide class of optimization problems can be solved in polynomial time on H-graphs (Corollary 16).

▶ **Theorem 10.** *Every H-graph G has at most $(2|G|+1)^{\|H\|} + \|H\| \cdot (2|G|)^2$ minimal separators.*

Proof (sketch). Let G be a H-graph and let F be a subdivision of H where G can be represented as the intersection graph of $\{M_v,\ v \in V(G)\}$. For every subset $V \subseteq V(G)$, the *border edges* of V are the edges of F with one endpoint in M_V and one endpoint in $V(F) \setminus M_V$. Let R be the union of border edges over $\{M_v,\ v \in V(G)\}$. Observe that for every $V \subseteq V(G)$, the set of border edges of V is a subset of R. For every $S \subseteq E(F)$, let V_S be the set of all vertices of G whose model contain some edge of S.

▶ **Claim 11** (★). *For every minimal separator X in G, there is a $S \subseteq R$ such that $X = V_S$.*

From Claim 11 we can already deduce that the number of minimal separators of G is at most the number of subsets of R. To get better bounds, we need other observations. The next claim follows from the fact that $F[M_V]$ is connected.

[6] We refer the reader to [21] for an accurate definition of these problems.

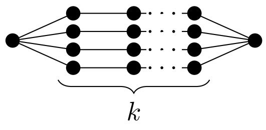


Figure 1 A θ_4-graph.

▶ **Claim 12.** *For every $V \subseteq V(G)$ such that M_V induces a connected subgraph of F, and every $e \in E(H)$, the set M_V has at most two border edges in $E(P_e)$. Hence, $|R| \leq 2|G| \cdot \|H\|$.*

▶ **Claim 13.** *For every minimal separator X of G, if $S \subseteq R$ is the subset of edges of F defined in the proof of Claim 11, then either $|S| = 2$ and $S \subseteq E(P_e)$ for some $e \in E(H)$, or $|S \cap E(P_e)| \leq 1$ for every $e \in E(H)$.*

Proof (sketch). Let A, B be two connected components of $G \setminus X$ such that $N(A) = N(B) = X$. From Claim 12 we get $|S \cap E(P_e)| \leq 2$ for every $e \in E(H)$. Intuitively, if $|S \cap E(P_e)| = 2$ for some $e \in E(H)$ then one of M_A and M_B contains only interior vertices of P_e. From the definition of S we deduce $|S| = 2$. ◀

Therefore, for every minimal separator X of G, there is a set $S \subseteq R$ such that:
1. either $|S \cap E(P_e)| \leq 1$ for every $e \in E(H)$;
2. or $|S| = 2$ and $S \subseteq E(P_e)$ for some $e \in E(H)$;

In order to upper-bound the number of possible minimal separators of G, we can consequently upper-bound the number of sets $S \subseteq R$ that satisfy one of the two conditions above, and (using Claim 12) obtain the bound of $(2|G| + 1)^{\|H\|} + \|H\| \cdot (2|G|)^2$. ◀

For every $r \in \mathbb{N}$, let θ_r be the graph with 2 vertices and r parallel edges. Lemma 14 shows that the exponential contribution of $\|H\|$ in Theorem 10 cannot be avoided. Figure 1 shows an example of a θ_4-graph as in its proof, with at least k^4 minimal separators.

▶ **Lemma 14 (★).** *For every $r \in \mathbb{N}$, there is a θ_r-graph G with at least $\left(\frac{|G|-2}{r}\right)^r$ minimal separators.*

Algorithmic applications. Let us consider the following generic problem described in [14].

Optimal Induced Subgraph for $\mathcal{P}$ and t (OIS($\mathcal{P}, t$) for short)
Input: A graph G;
Task: Find sets $X \subseteq F \subseteq V(G)$ such that X is of maximum size, the induced subgraph $G[F]$ is of treewidth at most t, and $\mathcal{P}(G[F], X)$ is true.

Fomin, Todinca, and Villanger proved that when the property $\mathcal{P}$ can be expressed in Counting Monadic Second Order logic (CMSOL, see [14]), the problem OIS($\mathcal{P}, t$) can be easily solved on classes of graphs that have a polynomial number of minimal separators. This includes natural optimization problems like Treewidth, Feedback Vertex Set, Maximum Induced Subgraph excluding a planar minor, and various packing problems.

▶ **Theorem 15** ([14]). *For any fixed t and CSMO property $\mathcal{P}$, OIS($\mathcal{P}, t$) on an n-vertex graph G with s minimal separators, is solvable in time $\mathcal{O}(s^2 \cdot n^{t+4} \cdot f(t, \mathcal{P}))$, for some function f of t and $\mathcal{P}$ only. In particular, the problem is solvable in polynomial time for classes of graphs whose number of minimal separator is upper-bounded by a polynomial function of their order.*

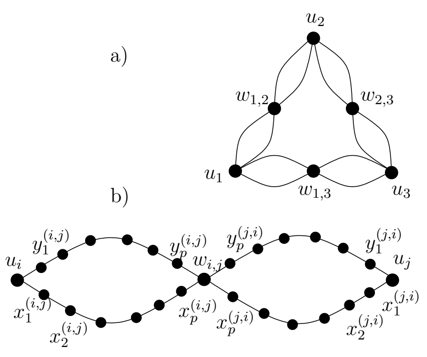


Figure 2 The construction of H for $k = 3$ and the subdivision of its edges.

We deduce the following result about H-graphs.

▶ **Corollary 16.** *Let H be a graph. For any fixed t and CSMO property $\mathcal{P}$, $\mathrm{OIS}(\mathcal{P}, t)$ can be solved in polynomial time $\mathcal{O}(n^{\mathcal{O}(|V(H)|)} \cdot n^{t+4} \cdot f(t, \mathcal{P}))$ on H-graphs.*

4 Parameterized complexity of basic problems for H-graphs

In this section we investigate the parameterized complexity of some basic graph problems for H-graphs: Dominating Set, Independent Set and Clique.

4.1 Hardness of of Dominating Set and Independent Set for H-graphs

Recall that Dominating Set and Independent Set, given a graph G and a positive integer k, ask whether G has a dominating set of size at most k and independent set of size at least k respectively. In this section we prove W[1]-hardness of Dominating Set and Independent Set for H-graphs (Theorem 17). Our proofs use a reduction from the Multicolored Clique problem. This problem, given a graph G and a k-partition $V_1, \dots, V_k$ of $V(G)$, asks whether G has a k-clique with exactly one vertex in each V_i for $i \in \{1, \dots, k\}$. The problem is well-known to be W[1]-complete when parameterized by k [12, 23].

▶ **Theorem 17.** *Dominating Set and Independent Set are W[1]-hard for H-graphs when parameterized by $k + \|H\|$, even if an H-representation of G is given.*

Proof (sketch). Let us show the W[1]-hardness for Independent Set. Let $(G, V_1, \dots, V_k)$ be an instance of Multicolored Clique. We assume that $k \geq 2$ and $|V_i| = p$ for $i \in \{1, \dots, k\}$. Denote by $v_1^i, \dots, v_p^i$ the vertices of V_i for $i \in \{1, \dots, k\}$.

We construct the multigraph H as follows:

(i) Construct k nodes $u_1, \dots, u_k$.

(ii) For every $1 \leq i < j \leq k$, construct a node $w_{i,j}$ and two pairs of parallel edges $u_i w_{i,j}$ and $u_j w_{i,j}$.

(See Figure 2 a).) Note that $|V(H)| = k(k+1)/2$ and $|E(H)| = 2k(k-1)$.

Then we construct the subdivision H' of H obtained by subdividing each edge p times. We denote the subdivision nodes for 4 edges of H constructed for each pair $0 \leq i < j \leq k$ in (ii) by $x_1^{(i,j)}, \dots, x_p^{(i,j)}$, $y_1^{(i,j)}, \dots, y_p^{(i,j)}$, $x_1^{(j,i)}, \dots, x_p^{(j,i)}$ and $y_1^{(j,i)}, \dots, y_p^{(j,i)}$ as it is shown in Figure 2 b).

To simplify notations, we assume that $u_i = x_0^{(i,j)} = y_0^{(i,j)}$, $u_j = x_0^{(j,i)} = y_0^{(j,i)}$ and $w_{i,j} = x_{p+1}^{(i,j)} = y_{p+1}^{(i,j)} = x_{p+1}^{(j,i)} = y_{p+1}^{(j,i)}$. Now we construct the H-graph G' by defining its

H-representation $\mathcal{M} = \{M_v\}_{v\in V(G')}$ where the model of each vertex is a connected subset of $V(H')$. Recall that G is the graph of the original instance of Multicolored Clique.

1. For each $i \in \{1,\dots,k\}$ and $s \in \{1,\dots,p\}$, construct a vertex z^i_s with the model
$$M_{z^i_s} = \cup_{j\in\{1,\dots,k\},j\neq i}(\{x^{(i,j)}_0,\dots,x^{(i,j)}_{s-1}\} \cup \{y^{(i,j)}_0,\dots,y^{(i,j)}_{p-s}\}).$$
2. For each edge $v^i_s v^j_t \in E(G)$ for $s,t \in \{1,\dots,p\}$ and $1 \le i < j \le k$, construct a vertex $r^{(i,j)}_{s,t}$ with the model
$$M_{r^{(i,j)}_{s,t}} = (\{x^{(i,j)}_s,\dots,x^{(i,j)}_{p+1}\}) \cup (\{y^{(i,j)}_{p-s+1},\dots,y^{(i,j)}_{p+1}\}) \cup (\{x^{(j,i)}_t,\dots,x^{(j,i)}_{p+1}\}) \cup (\{y^{(j,i)}_{p-t+1},\dots,y^{(j,i)}_{p+1}\}).$$

Finally, we define $k' = k(k+1)/2$. We claim that $(G, V_1,\dots,V_k)$ is a yes-instance of Multicolored Clique if and only if G' has an independent set of size k'. The proof is based on the following crucial property of our construction, that can be easily checked.

▶ **Claim 18.** *For every $0 \le i < j \le k$, a vertex $z^i_h \in V(G')$ (a vertex $z^j_h \in V(G')$) is not adjacent to a vertex $r^{(i,j)}_{s,t} \in V(G')$ corresponding to the edge $v^i_s v^j_t \in E(G)$ if and only if $h = s$ ($h = t$, respectively).*

Let $\{v^1_{h_1},\dots,v^k_{h_k}\}$ be a clique of G. Consider the set $I = \{z^1_{h_1},\dots,z^k_{h_k}\} \cup \{r^{(i,j)}_{h_i,h_j} \mid 0 \le i < j \le k\}$ of vertices of G'. It is straightforward to verify using Claim 18 that I is an independent set of size k' in G'. Suppose now that G' has an independent set I of size k'. For each $i \in \{1,\dots,k\}$, the set $Z_i = \{z^i_h \mid 1 \le h \le p\}$ is a clique of G', and for each $1 \le i < j \le k$, the set $R_{i,j} = \{r^{(i,j)}_{s,t} \mid 1 \le s,t \le p, v^i_s v^j_t \in E(G)\}$ is also a clique of G'. Since all these $k + \binom{k}{2} = k(k+1)/2 = k'$ cliques form a partition of $V(G')$, we have that for each $i \in \{1,\dots,k\}$, there is a unique $z^i_{h_i} \in Z_i \cap I$, and for every $1 \le i < j \le k$, there is a unique $r^{(i,j)}_{s_i,s_j} \in R_{i,j} \cap I$. Since $r^{(i,j)}_{s_i,s_j}$ is not adjacent to $z^i_{h_i}$ and $z^j_{h_j}$, we obtain that $s_i = h_i$ and $s_j = h_j$ by Claim 18. It implies that $v^i_{h_i} v^j_{h_j} \in E(G)$. Since it holds for every $1 \le i < j \le k$, $\{v^1_{h_1},\dots,v^k_{h_k}\}$ is a clique in G.

This completes the W[1]-hardness proof for Independent Set. Our proof can be modified to show the W[1]-hardness of Dominating Set. ◀

We proved in Theorem 2 that for every fixed H, every H-graph has mim-width at most $2\|H\|$. We deduce from the negative results above the following corollary.

▶ **Corollary 19.** Dominating Set *and* Independent Set *are* W[1]*-hard when parameterized by the solution size plus the mim-width of the input.*

We note that the construction in the proof of Theorem 17 has been adapted in [19] to show that the Feedback Vertex Set problem is W[1]-hard on H-graphs when parameterized by the solution size plus the number of edges of H.

4.2 Dominating Set for T-graphs

In this section we show that Dominating Set is FPT for chordal graphs when the problem is parameterized by the *leafage* of the input graph, that is, by the minimum number of leaves in a clique tree for the input graph. This result is somehow tight since Dominating Set is well-known to be W[2]-hard for split graphs when parameterized by the solution size [24]. Recall also that Independent Set is polynomial-time solvable for chordal graphs [15, 17] and, therefore, for H-graphs if H is a tree.

Let G be a graph. As it is standard, we say that $u \in V(G)$ (resp. $D \subseteq V(G)$) *dominates* $v \in V(G)$ if $v \in N_G[u]$ (resp. $v \in N_G[D]$) and u (resp. D) dominates a set $W \subseteq V(G)$ if every vertex of W is dominated by u (resp. some vertex of D). Let $\mathcal{K}$ be the set of (inclusion-wise) maximal cliques of G and let $\mathcal{K}_v \subseteq \mathcal{K}$ be the set of maximal cliques containing $v \in V(G)$. A tree T whose node set is $\mathcal{K}$ such that $\mathcal{K}_v$ induce a subtree of T for every $v \in V(G)$ is called a *clique tree* of G. It is well-known [16] that G is a chordal graph if and only if G has a clique tree T. Moreover, if T is a clique tree of G, then G is an intersection graph of subtrees of T, that is, G is a T-graph. Note that a clique tree of a chordal graph is not necessarily unique. For a connected chordal graph G, the *leafage* $\ell(G)$ of G is the minimum number of leaves in its clique tree. It was shown by Habib and Stacho in [18] that given a connected chordal graph G, we can construct in polynomial time its clique tree T with $\ell(G)$ leaves and a T-representation of G.

Let T be a tree and let G be a connected T-graph with its T-representation $\mathcal{M} = \{M_v\}_{v \in V(G)}$ with respect to a subdivision T' of T. For nonempty $U \subseteq V(T)$, we say that $v \in V(G)$ is a *U-vertex* if $M_v \cap V(T) = U$. If $U = \{u\}$, we write u-vertex instead of $\{u\}$-vertex. We denote the set of U-vertices by $V_G(U)$ and $V_G(u)$ if $U = \{u\}$. We also denote by $V_G(T)$ the set of all U-vertices of G for all nonempty $U \subseteq V(T)$. For $e \in E(T)$, $v \in V(G)$ is an e-vertex if M_v contains only subdivision nodes of T' from the path in T' corresponding to e in T. The set of e-vertices is denoted by $V_G(e)$.

We use the following lemma to upper bound the number of vertices in a minimum dominating set whose models contain given nodes of T.

▶ **Lemma 20** (★). *Let D be a minimum dominating set of G. Let $X \subseteq V(T)$ be an inclusion maximal set of nodes of T such that i) for every $x \in X$, there is $u \in D$ with $x \in M_u$ and ii) for every $xy \in E(T)$ with $x, y \in X$, there is $u \in D$ with $x, y \in M_u$. Then the set $U = \{u \in D \mid X \cap M_u \neq \emptyset\}$ contains at most $|N_T[X]|$ vertices.*

In particular, since $|N_T(X)|$ is at most the number of leaves ℓ, we have that $|U| \leq |X| + \ell$. We use Lemma 20 to obtain an upper bound for the number of vertices in a minimum dominating set whose models contain nodes of T. The next lemma is crucial for our algorithm as it allows to restrict the choice of vertices in a dominating set whose models contain branching vertices. Observe that the models of other vertices form a union of disjoint interval graphs.

▶ **Lemma 21** (★). *Let D be a minimum dominating set of G. Then $|D \cap V_G(T)| \leq 3|V(T)| - 2$.*

We consider the following auxiliary problem for T-graphs.

Dominating Set Extension
Input: A tree T and a graph G with a given T-representation, positive integers k and d, a *labeling* function $c\colon \bigcup_{x \in V(T)} V_G(x) \to \mathbb{N}$, and a collection of sets $\{C_x\}_{x \in V(T)}$ of size at most d where each $C_x \subseteq c(V_G(x))$ (some sets could be empty) such that for every dominating set D of G of minimum size with the properties that
1. D has at most d x-vertices for $x \in V(T)$, and
2. for each $x \in V(T)$, $C_x \subseteq c(D \cap V_G(x))$,

it holds that the number of nodes $x \in V(T)$ such that D contains an x-vertex is maximum and for each $x \in V(T)$, $C_x = c(D \cap V_G(x))$.
Task: Decide whether there is a dominating set D' of G of size at most k containing at most d x-vertices for $x \in V(T)$ such that for each $x \in V(T)$, $C_x = c(D' \cap V_G(x))$.

Note that Dominating Set Extension is a *promise* problem: we are promised that there is D with the described properties but D itself is not given. Moreover, the promise could be false but we are not asked to verify it. The labeling c in the statement of the problem and the promise define, in fact, the choice of the vertices in a dominating set whose models contain branching vertices.

We use dynamic programming to solve this problem, but we are solving it only for graphs with special representations. Let $\mathcal{M} = \{M_v\}_{v\in V(G)}$ be a T-representation of G with respect to a subdivision T'. We say that $\mathcal{M}$ is *nice* if $|M_v \cap V(T)| \leq 1$ for $v \in V(G)$, i.e., each set M_v contains at most one branching node of T'. This considerably simplifies handling of the vertices whose models contain branching nodes that are selected to be included in a dominating set by our dynamic programming algorithm. As it is standard for dynamic programming, we pick a root r in T that defines the parent-child relation on $V(T)$ and $V(T')$.

▶ **Lemma 22** (★). *Given a nice r-rooted T-representation of the input graph where T is a tree with ℓ leaves,* Dominating Set Extension *can be solved in time $2^{\mathcal{O}((d+\ell)\log d)}n^{\mathcal{O}(1)}$. Moreover, it can be done by an algorithm that either returns a correct yes-answer or (possible incorrect) no-answer even if the promise is false.*

To be able to make a given T-representation nice, we define contractions of edges of T that transforms G as well. For an edge $e \in E(T)$, we say that G' is obtained *by contracting e in T* if G' is the (T/e)-graph with the model obtained as follows:

1. contract xy in T and, respectively, the (x, y)-path P in T', and denote the node obtained from x and y by z,
2. delete all e-vertices of G,
3. for each remaining vertex $u \in V(G)$, delete from M_u the subdivision nodes of P and replace x and y by z if at least one of these nodes is in M_u.

Note that $V(G') \subseteq V(G)$ and $G[V(G')]$ is a subgraph of G' but not necessarily induced since two vertices of G' that are not adjacent in G could be adjacent in G'.

Now we are ready to explain how we solve Dominating Set for chordal graphs of bounded leafage.

▶ **Theorem 23.** Dominating Set *can be solved in time $2^{\mathcal{O}(\ell^2)} \cdot n^{\mathcal{O}(1)}$ for connected chordal graphs with leafage at most ℓ.*

Proof (sketch). Let (G, k) be an instance of Dominating Set where G is a connected chordal graph.

We use the algorithm of Habib and Stacho [18] to compute its leafage $\ell(G)$. If $\ell(G) > \ell$, we stop and return a no-answer. Otherwise, we consider the clique tree T' of G constructed by the algorithm. We construct the tree T from T' by *dissolving* nodes of degree two, that is, for a node x of degree two with the neighbors y and z, we delete x and make y and z adjacent. Observe that since T is a tree with at most ℓ leaves that has no node of degree two, $|V(T)| \leq 2\ell - 2$. We have that G is a T-graph. Note also that the algorithm of Habib and Stacho [18] gives us a T-representation $\mathcal{M} = \{M_v\}_{v\in V(G)}$ where $M_v \in V(T')$ for $v \in V(G)$.

We consider $2^{|V(T)|} - 1 \leq 2^{2\ell-2} - 1$ nonempty subsets of $V(T)$ and construct a *coloring* $c\colon V_G(T) \to \{1, \ldots, 2^{|V(T)|}\}$ such that for $u, v \in V_G(T)$, $c(u) = c(v)$ if and only if u and v are U-vertices for the same $U \subseteq V(T)$.

By Lemma 21, a minimum dominating set of G contains at most $3|V(T)| - 2 \leq 6\ell - 8$ vertices of $V_G(T)$. Clearly, these vertices can have at most $6\ell - 8$ distinct colors. We consider all sets $C \subseteq \{1, \ldots, 2^{|V(T)|}\}$ of distinct colors of size at most $6\ell - 8$ and for each C, we aim to find a minimum dominating set of G whose vertices in $V_G(T)$ are colored by the maximum number of distinct colors and are colored exactly by the colors of C. Since we consider all possible choices of C, it holds for some C. Toward this aim, we apply the following rule.

Rule 1. If there is an xy-vertex w of G for $xy \in E(T)$ such that i) $x, y \notin M_u$ for $u \in V_G(T)$ with $c(u) \in C$ and ii) there is $v \in V_G(T)$ such that $x, y \in M_v$, then discard the choice of C.

Now we are looking for a dominating set D of minimum size such that $c(D \cap V_G(T)) = C$. We use the following rule.

Rule 2. If there is a U-vertex u of G for nonempty $U \subseteq V(T)$ such that i) $c(u) \notin C$ and ii) there is $c \in C$ such that for every $v \in V_G(T)$ with $c(v) = c$, v dominates u, then delete u.

Our next aim is to construct a nice representation. Let

$$\begin{aligned} A &= \{xy \in E(T) \mid x, y \in M_u \text{ for some } u \in V_G(T) \text{ such that } c(u) \in C\}, \\ A' &= \{xy \in E(T) \mid x, y \in M_u \text{ for some } u \in V_G(T) \text{ such that } c(u) \notin C \text{ and} \\ &\qquad\qquad x, y \notin M_v \text{ for } v \in V_G(T) \text{ such that } c(v) \in C\}. \end{aligned}$$

Because of Rule 1, there is no e-vertex for $e \in A'$. We contract the edges $e \in A \cup A'$. Let $\hat{T}$ (resp. $\hat{T}'$) be the tree obtained from T (resp. T') by contracting the paths that correspond to the contracted edge. We also construct the graph $\hat{G}$ that is obtained from G by contracting these edges of T and we also construct its $\hat{T}$-representation $\hat{\mathcal{M}} = \{\hat{M}_v\}_{v \in V(\hat{G})}$ where $\hat{M}_v \in V(\hat{T}')$ for $v \in V(\hat{G})$. We set $\hat{c} = c|_{V(\hat{G})}$ and let $C_x = \{c,\ \exists u \in V_{\hat{G}}(x) \text{ s.t. } \hat{c}(u) = c\}$ for $x \in V(\hat{T})$. Observe that $\hat{\mathcal{M}}$ is a nice $\hat{T}$-representation of $\hat{G}$. Indeed, for every $xy \in E(T)$ such that $x, y \in M_u$ for $u \in V_G(T)$ we have that $xy \in A$ if $c(u) \in C$ and $xy \in A'$ if $c(u) \notin C$ because of Rule 2, and all such edges xy are contracted.

We show that the contraction of the edges of $A \cup A'$ is safe in the sense that D is a minimum dominating set with $C = c(D \cap V_G(T))$ if and only if D is a minimum dominating set of G'such that $C = \hat{c}(D \cap V_{\hat{G}}(\hat{T}))$. Note that the condition $C = \hat{c}(D \cap V_{\hat{G}}(\hat{T}))$ is equivalent to the condition that for every $x \in V(\hat{T})$, $C_x = \hat{c}(D \cap V_{\hat{G}}(x))$, because the $\hat{T}$-representation of $\hat{G}$ is nice and $C_x \cap C_y = \emptyset$ for distinct $x, y \in V(\hat{T})$. We set $d = |V(T)| + \ell \leq 3\ell - 2$ and apply the next rule.

Rule 3. If there is $x \in V(\hat{T})$ with $|C_x| > d$, then discard the current choice of C.

By Lemma 20, we have that if a set of nodes X of T is contracted into a single vertex x of $\hat{T}$, then D has at most $|X| + \ell$ vertices whose models contain a vertex of X and, therefore, the number of vertices colored by the colors of C_x in D is at most d.

We select arbitrarily a node r to be the root of $\hat{T}$ and $\hat{T}'$ respectively. Then we apply Lemma 22 for the instance $(\hat{T}, k, d, \hat{c}, \{C_x\}_{x \in V(\hat{T})})$ of Dominating Set Extension.

Recall that Dominating Set Extension is a promise problem. If the algorithm from Lemma 22 returns a yes-answer, it means that there is a dominating set D of $\hat{G}$ of size at most k such that for each $x \in V(\hat{T})$, $C_x = c(D \cap V_{\hat{G}}(x))$. It means that the input graph G has a dominating set of size at most k. Still, if the promise is false, the algorithm can return an incorrect no-answer. Recall that the promise of Dominating Set Extension is the following: for every dominating set D of $\hat{G}$ of minimum size with the properties that

1. D has at most d x-vertices for $x \in V(\hat{T})$, and
2. for each $x \in V(\hat{T})$, $C_x \subseteq c(D \cap V_G(\hat{x}))$,

it holds that the number of nodes $x \in V(\hat{T})$ such that D contains an x-vertex is maximum and for each $x \in V(\hat{T})$, $C_x = c(D \cap V_G(\hat{x}))$. We prove that if C is chosen in such a way that G has a minimum dominating set D that has the maximum number of vertices of $V_G(T)$ and whose vertices in $V_G(T)$ are colored exactly by the colors of C, then this promise holds for the corresponding instance of Dominating Set Extension constructed for this choice of C. Therefore, if (G, k) is a yes-instance of Dominating Set, then for some choice of C, we obtain a yes-answer. ◀

The theorem immediately gives the following corollary for T-graphs.

▶ **Corollary 24.** Dominating Set *can be solved in time* $2^{\mathcal{O}(|T|^2)} \cdot n^{\mathcal{O}(1)}$ *for* T*-graphs if* T *is a tree.*

4.3 A polynomial kernel for Clique

It was observed in [8] that the Clique problem is FPT for H-graphs when parameterized by the solution size k and $\|H\|$ (even when no H-representation of G is given). We show that Clique admits a polynomial kernel when a representation is given.

▶ **Theorem 25 (★).** *The* Clique *problem for* H*-graphs admits a kernel with at most* $(k-1)|V(H)|$ *vertices if an* H*-representation of the input graph is given.*

References

1 Rémy Belmonte and Martin Vatshelle. On graph classes with logarithmic boolean-width. *arXiv preprint*, 2010. arXiv:1009.0216.

2 Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoretical Computer Science*, 511:54–65, 2013. doi:10.1016/j.tcs.2013.01.011.

3 Anne Berry, Jean-Paul Bordat, and Olivier Cogis. *Generating All the Minimal Separators of a Graph*, pages 167–172. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999. doi:10.1007/3-540-46784-X_17.

4 M. Biró, M. Hujter, and Zs. Tuza. Precoloring extension. I. Interval graphs. *Discrete Mathematics*, 100(1):267–279, 1992. doi:10.1016/0012-365X(92)90646-W.

5 Vincent Bouchitté and Ioan Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM Journal on Computing*, 31(1):212–232, 2001. doi:10.1137/S0097539799359683.

6 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Boolean-width of graphs. *Theoretical Computer Science*, 412(39):5187–5204, 2011. doi:10.1016/j.tcs.2011.05.022.

7 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theoretical Computer Science*, 511:66–76, 2013. Exact and Parameterized Computation. doi:10.1016/j.tcs.2013.01.009.

8 Steven Chaplick, Martin Töpfer, Jan Voborník, and Peter Zeman. On H-topological intersection graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 167–179. Springer, 2017. arXiv:1608.02389.

9 Steven Chaplick and Peter Zeman. Combinatorial problems on H-graphs. In *The European Conference on Combinatorics, Graph Theory and Applications (EUROCOMB'17), Electronic Notes in Discrete Mathematics*, volume 61, pages 223–229, 2017. arXiv:1706.00575. doi:10.1016/j.endm.2017.06.042.

10 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Dániel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

11 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

12 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009. doi:10.1016/j.tcs.2008.09.065.

13 Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the tractability of optimization problems on h-graphs. *CoRR*, abs/1709.09737, 2017. arXiv:1709.09737.

14 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM J. Comput.*, 44(1):54–87, 2015. arXiv:1309.1559. doi:10.1137/140964801.

15 Fănică Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM J. Comput.*, 1(2):180–187, 1972. doi:10.1137/0201013.

16 Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Combinatorial Theory Ser. B*, 16:47–56, 1974.

17 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57 of *Annals of Discrete Mathematics*. Elsevier Science B.V., Amsterdam, second edition, 2004. With a foreword by Claude Berge.

18 Michel Habib and Juraj Stacho. Polynomial-time algorithm for the leafage of chordal graphs. In *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 290–300. Springer, 2009. doi:10.1007/978-3-642-04128-0_27.

19 L. Jaffke, O. Kwon, and J. A. Telle. A note on the complexity of Feedback Vertex Set parameterized by mim-width. *arXiv preprint*, 2017. arXiv:1711.05157. arXiv:1711.05157.

20 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. A note on the complexity of feedback vertex set parameterized by mim-width. *CoRR*, abs/1711.05157, 2017. arXiv:1711.05157.

21 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Polynomial-time algorithms for the longest induced path and induced disjoint paths problems on graphs of bounded mim-width. *arXiv preprint*, 2017. arXiv:1708.04536.

22 Ton Kloks, H Bodlaender, Haiko Müller, and Dieter Kratsch. Computing treewidth and minimum fill-in: All you need are the minimal separators. *Algorithms—ESA'93*, pages 260–271, 1993.

23 Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. Comput. Syst. Sci.*, 67(4):757–771, 2003. doi:10.1016/S0022-0000(03)00078-3.

24 Venkatesh Raman and Saket Saurabh. Short cycles make W[1]-hard problems hard: FPT algorithms for W[1]-hard problems in graphs with no short cycles. *Algorithmica*, 52(2):203–225, 2008. doi:10.1007/s00453-007-9148-9.

On the Optimality of Pseudo-polynomial Algorithms for Integer Programming

Fedor V. Fomin
Department of Informatics, University of Bergen, Norway
fomin@ii.uib.no

Fahad Panolan
Department of Informatics, University of Bergen, Norway
fahad.panolan@ii.uib.no

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

Saket Saurabh
Institute of Mathematical Sciences, HBNI, Chennai, India and University of Bergen, Norway
saket@imsc.res.in

Abstract

In the classic *Integer Programming* (IP) problem, the objective is to decide whether, for a given $m \times n$ matrix A and an m-vector $b = (b_1, \dots, b_m)$, there is a non-negative integer n-vector x such that $Ax = b$. Solving (IP) is an important step in numerous algorithms and it is important to obtain an understanding of the precise complexity of this problem as a function of natural parameters of the input.

The classic pseudo-polynomial time algorithm of Papadimitriou [J. ACM 1981] for instances of (IP) with a constant number of constraints was only recently improved upon by Eisenbrand and Weismantel [SODA 2018] and Jansen and Rohwedder [ArXiv 2018]. We continue this line of work and show that under the Exponential Time Hypothesis (ETH), the algorithm of Jansen and Rohwedder is nearly optimal. We also show that when the matrix A is assumed to be non-negative, a component of Papadimitriou's original algorithm is already nearly optimal under ETH.

This motivates us to pick up the line of research initiated by Cunningham and Geelen [IPCO 2007] who studied the complexity of solving (IP) with non-negative matrices in which the number of constraints may be unbounded, but the branch-width of the column-matroid corresponding to the constraint matrix is a constant. We prove a lower bound on the complexity of solving (IP) for such instances and obtain optimal results with respect to a closely related parameter, path-width. Specifically, we prove *matching* upper and lower bounds for (IP) when the *path-width* of the corresponding column-matroid is a constant.

2012 ACM Subject Classification Theory of computation $\rightarrow$ Integer programming

Keywords and phrases Integer Programming, Strong Exponential Time Hypothesis, Branch-width of a matrix, Fine-grained Complexity

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.31

Related Version A full version of the paper is available at `https://arxiv.org/abs/1607.05342`.

© Fedor V. Fomin, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh;
licensed under Creative Commons License CC-BY

1 Introduction

In the classic *Integer Programming* problem, the input is an $m \times n$ integer matrix A, and an m-vector $b = (b_1, \dots, b_m)$. We consider the feasibility version of the problem, where the objective is to find a non-negative integer n-vector x (if one exists) such that $Ax = b$. Solving this problem, denoted by (IP), is a fundamental step in numerous algorithms and it is important to obtain an understanding of the precise complexity of this problem as a function of natural parameters of the input. Throughout the paper we denote Δ for the largest absolute value of the entries of A.

(IP) is known to be NP-hard. However, there are two classic algorithms due to Lenstra [13] and Papadimitriou [16] solving (IP) in polynomial or pseudo-polynomial time for two important cases when the number of variables and the number of constraints are bounded. These algorithms in some sense complement each other.

The algorithm of Lenstra shows that (IP) is solvable in polynomial time when the number of variables is bounded. Actually, the result of Lenstra is even stronger: (IP) is *fixed-parameter tractable* parameterized by the number of variables. However, the running time of Lenstra's algorithm is doubly exponential in n. Later, Kannan [12] provided an algorithm for (IP) running in time $n^{\mathcal{O}(n)}$. Deciding whether the running time $n^{\mathcal{O}(n)}$ can be improved to $2^{\mathcal{O}(n)}$ is a long-standing open question.

Our work is motivated by the complexity analysis of the complementary case when the number of constraints is bounded. (IP) is NP-hard already for $m = 1$ (the Knapsack problem) but solvable in pseudo-polynomial time. In 1981, Papadimitriou [16] extended this result by showing that (IP) is solvable in pseudo-polynomial time on instances for which the number of constraints m is a constant. The algorithm of Papadimitriou consists of two steps. The first step is combinatorial, showing that if the entries of A and b are from $\{0, \pm 1, \dots, \pm d\}$, and (IP) has a solution, then there is also a solution which is in $\{0, 1, \dots, n(md)^{2m+1}\}^n$. The second, algorithmic step shows that if (IP) has a solution with the maximum entry at most B, then the problem is solvable in time $\mathcal{O}((nB)^{m+1})$. Thus the total running time of Papadimitriou's algorithm is $\mathcal{O}(n^{2m+2} \cdot (md)^{(m+1)(2m+1)})$, where $d = \max\{\Delta, \|b\|_\infty\}$. There was no algorithmic progress on this problem until the very recent breakthrough of Eisenbrand and Weismantel [6]. They proved the following result.

▶ **Proposition 1** (Theorem 2.2, Eisenbrand and Weismantel [6]). *(IP) with $m \times n$ matrix A is solvable in time $(m \cdot \Delta)^{\mathcal{O}(m)} \cdot \|b\|_\infty^2$.*

Then, Jansen and Rohwedder improved Proposition 1 and gave a matching lower bound very recently [10].

▶ **Proposition 2** (Jansen and Rohwedder [10]). *(IP) with $m \times n$ matrix A is solvable in time $\mathcal{O}(m\Delta)^m \log(\Delta) \log(\Delta + \|b\|_\infty)$. Assuming the Strong Exponential Time Hypothesis (SETH), there is no algorithm for (IP) running in time $n^{\mathcal{O}(1)} \cdot \mathcal{O}(m(\Delta + \|b\|_\infty))^{m-\delta}$ for any $\delta > 0$.*

SETH is the hypothesis that CNF-SAT cannot be solved in time $(2 - \epsilon)^n m^{\mathcal{O}(1)}$ on n-variable m-clause formulas for any constant ϵ. ETH is the hypothesis that 3-SAT cannot be solved in time $2^{o(n)}$ on n-variable formulas. Both ETH and SETH were first introduced in the work of Impagliazzo and Paturi [8], which built upon earlier work of Impagliazzo, Paturi and Zane [9]. One of the natural question is whether the exponential dependence of $\|b\|_\infty$ can be improved significantly at the cost of super polynomial dependence on n. Our first theorem provides a conditional lower bound indicating that any significant improvements are unlikely.

▶ **Theorem 3.** *Unless the Exponential Time Hypothesis (ETH) fails, (IP) with $m \times n$ matrix A cannot be solved in time $n^{o(\frac{m}{\log m})} \cdot \|b\|_\infty^{o(m)}$ even when the constraint matrix A is non-negative and each entry in any feasible solution is at most 2.*

Let us note that since the bound in Theorem 3 holds for a non-negative matrix A, we can always reduce (in polynomial time) the original instance of the problem to an equivalent instance where the maximum value Δ in the constraint matrix A does not exceed $\|b\|_\infty$. Thus Theorem 3 also implies the conditional lower bound $n^{o(\frac{m}{\log m})} \cdot (\Delta \cdot \|b\|_\infty)^{o(m)}$. When $m = \mathcal{O}(n)$, our bound also implies the lower bound $(n \cdot m)^{o(\frac{m}{\log m})} \cdot (\Delta \cdot \|b\|_\infty)^{o(m)}$. We complement Theorem 3 by turning our focus to the dependence of algorithms solving (IP) on m alone, and obtaining the following theorem.

▶ **Theorem 4.** *Unless ETH fails, (IP) with $m \times n$ matrix A cannot be solved in time $f(m) \cdot (n \cdot \|b\|_\infty)^{o(\frac{m}{\log m})}$ for any computable function f. The result holds even when the constraint matrix A is non-negative and each entry in any feasible solution is at most 1.*

Although Theorem 3 provides a better dependence on $\|b\|_\infty$, Theorem 4 provides much more information on how the complexity of the problem depends on m. Since several parameters are involved in this running time estimation, a natural objective is to study the possible tradeoffs between them. For instance, consider the $\mathcal{O}(m\Delta)^m \log(\Delta) \log(\Delta + \|b\|_\infty)$ time algorithm (Proposition 2) for (IP). A natural follow up question is the following. Could it be that by allowing a significantly worse dependence (a superpolynomial dependence) on n and $\|b\|_\infty$ and an *arbitrary* dependence on m, one might be able to improve the dependence on Δ alone? Theorem 4 provides a strong argument against such an eventuality. Indeed, since the lower bound of Theorem 4 holds even for non-negative matrices, it rules out algorithms with running time $f(m) \cdot \Delta^{o(\frac{m}{\log m})} \cdot (n \cdot \|b\|_\infty)^{o(\frac{m}{\log m})}$. Therefore, obtaining a subexponential dependence of Δ on m even at the cost of a superpolynomial dependence of n and $\|b\|_\infty$ on m, and an arbitrarily bad dependence on m is as hard as obtaining a subexponential algorithm for 3-SAT.

We now motivate our remaining results. It is straightforward to see that when the matrix A happens to be non-negative, the algorithm of Papadimitriou [16] runs in time $\mathcal{O}((n \cdot \|b\|_\infty)^{m+1})$. Due to Theorems 3 and 4, the dynamic programming step of the algorithm of Papadimitriou for (IP) when the maximum entry in a solution as well as in the constraint matrix is bounded, is already close to optimal. Consequently, any quest for "faster" algorithms for (IP) must be built around the use of additional structural properties of the matrix A. Cunningham and Geelen [1] introduced such an approach by considering the *branch decomposition* of the matrix A. They were motivated by the fact that the result of Papadimitriou can be interpreted as a result for matrices of constant *rank* and branch-width is a parameter which is upper bounded by rank plus one. For a matrix A, the *column-matroid* of A denotes the matroid whose elements are the columns of A and whose independent sets are precisely the linearly independent sets of columns of A. We postpone the formal definitions of branch decomposition and branch-width till the next section. For (IP) with a *non-negative* matrix A, Cunningham and Geelen [1] showed that when the branch-width of the column-matroid of A is constant, (IP) is solvable in pseudo-polynomial time.

▶ **Proposition 5** (Cunningham and Geelen [1]). *(IP) with a non-negative $m \times n$ matrix A given together with a branch decomposition of its column matroid of width k, is solvable in time $\mathcal{O}((\|b\|_\infty + 1)^{2k} mn + m^2 n)$.*

We analyze the complexity of (IP) parameterized by the branch-width of A, by making use of SETH.

▶ **Theorem 6.** *Unless SETH fails, (IP) with a non-negative $m \times n$ constraint matrix A cannot be solved in time $f(\mathsf{bw})(\|b\|_\infty + 1)^{(1-\epsilon)\mathsf{bw}}(mn)^{\mathcal{O}(1)}$ or $f(\|b\|_\infty)(\|b\|_\infty + 1)^{(1-\epsilon)\mathsf{bw}}(mn)^{\mathcal{O}(1)}$, for any computable function f. Here* bw *is the branchwidth of the column matroid of A.*

In recent years, SETH has been used to obtain several tight conditional bounds on the running time of algorithms for various optimization problems on graphs of bounded treewidth [14]. In fact, Theorem 6 follows from stronger lower bounds we prove using the path-width of A as our parameter of interest instead of the branch-width. The parameter *path-width* is closely related to the notion of *trellis-width* of a linear code, which is a parameter commonly used in coding theory [7]. For a matrix $A \in \mathbb{R}^{m \times n}$, computing the path-width of the column matroid of A is equivalent to computing the trellis-width of the linear code generated by A. Roughly speaking, the path-width of the column matroid of A is at most k, if there is a permutation of the columns of A such that in the matrix A' obtained from A by applying this column-permutation, for every $1 \leq i \leq n-1$, the *dimension* of the subspace of $\mathbb{R}^m$ obtained by taking the intersection of the subspace of $\mathbb{R}^m$ spanned by the first i columns with the subspace of $\mathbb{R}^m$ spanned by the remaining columns, is at most $k-1$.

The value of the parameter path-width is always at least the value of branch-width and thus Theorem 6 follows from the following theorems.

▶ **Theorem 7.** *Unless SETH fails, (IP) with even a non-negative $m \times n$ constraint matrix A cannot be solved in time $f(k)(\|b\|_\infty + 1)^{(1-\epsilon)k}(mn)^{\mathcal{O}(1)}$ for any computable function f and $\epsilon > 0$, where k is the path-width of the column matroid of A.*

▶ **Theorem 8.** *Unless SETH fails, (IP) with even a non-negative $m \times n$ constraint matrix A cannot be solved in time $f(\|b\|_\infty)(\|b\|_\infty + 1)^{(1-\epsilon)k}(mn)^{\mathcal{O}(1)}$ for any computable function f and $\epsilon > 0$, where k is the path-width of the column matroid of A.*

Although the proofs of both lower bounds have a similar structure, we believe that there are sufficiently many differences in the proofs to warrant stating and proving them separately.

Note that although there is still a gap between the upper bound of Cunningham and Geelen from Proposition 5 and the lower bound provided by Theorem 6, the lower bounds given in Theorems 8 and 7 are asymptotically tight in the following sense. The proof of Cunningham and Geelen in [1] actually implies the upper bound stated in Theorem 9. We provide a self-contained proof in the appended full version of the paper for the reader's convenience.

▶ **Theorem 9.** *(IP) with non-negative $m \times n$ matrix A given together with a path decomposition of its column matroid of width k is solvable in time $\mathcal{O}((\|b\|_\infty + 1)^{k+1}mn + m^2n)$.*

Then by Theorem 7, we cannot relax the $(\|b\|_\infty + 1)^k$ factor in Theorem 9 even if we allow in the running time an arbitrary function depending on k, while Theorem 8 shows a similar lower bound in terms of $\|b\|_\infty$ instead of k. Put together the results imply that no matter how much one is allowed to compromise on either the path-width or the bound on $\|b\|_\infty$, it is unlikely that the algorithm of Theorem 9 can be improved.

The path-width of matrix A does not exceed its rank and thus the number of constraints in (IP). Hence, similar to Proposition 5, Theorem 9 generalizes the result of Papadimitriou when restricted to non-negative matrices. Also we note that the assumption of non-negativity is unavoidable (without any further assumptions such as a bounded domain for the variables) in this setting because (IP) is NP-hard when the constraint matrix A is allowed to have negative values (in fact even when restricted to $\{-1, 0, 1\}$) and the branchwidth of the column matroid of A is at most 3. A close inspection of the instances they construct in their NP-hardness reduction shows that the column matroids of the resulting constraint matrices are in fact direct sums of circuits, implying that even their *path-width* is bounded by 3.

2 Preliminaries

We use $\mathbb{Z}_{\geq 0}$ and $\mathbb{R}$ to denote the set of non negative integers and real numbers, respectively. For any positive integer n, we use $[n]$ and $\mathbb{Z}_n$ to denote the sets $\{1,\ldots,n\}$ and $\{0,1,\ldots,n-1\}$, respectively. For convenience, we say that $[0]=\emptyset$. For any two vectors $b,b' \in \mathbb{R}^m$ and $i \in [m]$, we use $b[i]$ to denote the i^{th} coordinate of b and we write $b' \leq b$, if $b'[i] \leq b[i]$ for all $i \in [m]$. We often use 0 to denote the zero-vector whose length will be clear from the context. For a matrix $A \in \mathbb{R}^{m\times n}$, $I \subseteq [m]$ and $J \subseteq [n]$, $A[I,J]$ denote the submatrix of A obtained by the restriction of A to the rows indexed by I and columns indexed by J. The notion of the branch-width of graphs, and implicitly of matroids, was introduced by Robertson and Seymour in [17]. Let $M=(U,\mathcal{F})$ be a matroid with universe set U and family $\mathcal{F}$ of independent sets over U. We use r_M to denote the rank function of M. That is, for any $S \subseteq U$, $r_M(S)=\max_{S'\subseteq S, S'\in\mathcal{F}} |S'|$. For $X \subseteq U$, the *connectivity function* of M is defined as $\lambda_M(X)=r_M(X)+r_M(U\setminus X)-r_M(U)+1$.

For matrix $A \in \mathbb{R}^{m\times n}$, we use $M(A)$ to denote the column-matroid of A. In this case the connectivity function $\lambda_{M(A)}$ has the following interpretation. For $E=\{1,\ldots,n\}$ and $X \subseteq E$, we define $S(A,X)=\mathrm{span}(A|X)\cap\mathrm{span}(A|E\setminus X)$, where $A|X$ is the set of columns of A restricted to X and $\mathrm{span}(A|X)$ is the subspace of $\mathbb{R}^m$ spanned by the columns $A|X$. It is easy to see that the dimension of $S(A,X)$ is equal to $\lambda_{M(A)}(X)-1$.

A tree is *cubic* if its internal vertices all have degree 3. A *branch decomposition* of matroid M with universe set U is a cubic tree T and mapping μ which maps elements of U to leaves of T. Let e be an edge of T. Then the forest $T-e$ consists of two connected components T_1 and T_2. Thus every edge e of T corresponds to the partitioning of U into two sets X_e and $U\setminus X_e$ such that $\mu(X_e)$ are the leaves of T_1 and $\mu(U\setminus X_e)$ are the leaves of T_2. The *width* of edge e is $\lambda_M(X_e)$ and the width of branch decomposition (T,μ) is the maximum edge width, where maximum is taken over all edges of T. Finally, the *branch-width* of M is the minimum width taken over all possible branch decompositions of M.

The *path-width* of a matroid is defined as follows. Recall that a *caterpillar* is a tree which is obtained from a path by attaching leaves to some vertices of the path. Then the path-width of a matroid is the minimum width of a branch decomposition (T,μ), where T is a cubic caterpillar. Let us note that every mapping of elements of a matroid to the leaves of a cubic caterpillar corresponds to an ordering of these elements. Jeong, Kim, and Oum [11] gave a constructive fixed-parameter tractable algorithm to construct a path decomposition of width at most k for a column matroid of a given matrix.

For $q \geq 3$, let δ_q be the infimum of the set of constants c for which there exists an algorithm solving q-SAT with n variables and m clauses in time $2^{cn}\cdot m^{\mathcal{O}(1)}$. The *Exponential-Time Hypothesis (ETH)* and *Strong Exponential-Time Hypothesis (SETH)* are then formally defined as follows. ETH conjectures that $\delta_3>0$ and SETH that $\lim_{q\to\infty}\delta_q=1$.

3 ETH lower bounds on pseudopolynomial solvability of (IP)

In this section we prove Theorem 4. Here, we give a brief overview of the reduction and the intuition behind it. We use the ETH based lower bound result of Marx [15] for Partitioned Subgraph Isomorphism. For two graphs G and H, a map $\phi\colon V(G)\mapsto V(H)$ is called a *subgraph isomorphism* from G to H, if ϕ is injective and for any $\{u,v\}\in E(G)$, $\{\phi(u),\phi(v)\}\in E(H)$. In the Partitioned Subgraph Isomorphism problem, the input consists of two graphs G,H, a bijection $c_G\colon V(G)\mapsto[\ell]$ and a function $c_H\colon V(H)\mapsto[\ell]$, where $\ell=|V(G)|$ and the objective is to decide whether there a subgraph isomorphism ϕ from G to H such that for any $v\in V(G)$, $c_G(v)=c_H(\phi(v))$.

▶ **Lemma 10** ([15]). *If* Partitioned Subgraph Isomorphism *can be solved in time $f(G)n^{o(\frac{k}{\log k})}$, where f is an arbitrary function, $n = |V(H)|$ and $k = |E(G)|$, then ETH fails.*

To prove Theorem 4 we give a polynomial time reduction from Partitioned Subgraph Isomorphism to (IP) such that for every instance (G, H, c_G, c_H) of Partitioned Subgraph Isomorphism the reduction outputs an instance of (IP) where the constraint matrix has dimension $\mathcal{O}(|E(G)|) \times \mathcal{O}(|E(H)|)$ and the largest value in the target vector is $\max\{|E(H)|, |V(H)|\}$.

Let (G, H, c_G, c_H) be an instance of Partitioned Subgraph Isomorphism. Let $k = |E(G)|$ and $n = |V(H)|$. We construct an instance $Ax = b$ of (IP) from (G, H, c_G, c_H) in polynomial time. Without loss of generality we assume that $[n] = V(H)$ and that there are no isolated vertices in G. Hence, the number of vertices in G is at most $2k$. Let $m = |E(H)|$. For each $e \in E(H)$ we assign a unique integer from $[m]$. Let $\alpha\colon E(H) \mapsto [m]$ be the bijection which represents the assignment mentioned above. For any $i, j \in [\ell]$, we use $E_H(i, j)$ as a shorthand for the set of edges of H between $c_H^{-1}(i)$ and $c_H^{-1}(j)$. Finally, for ease of presentation we let $\{v_1, \ldots, v_\ell\} = V(G)$ and $c_G(v_i) = i$ for all $i \in [\ell]$, where $\ell = |V(G)|$.

We now formally define the (IP) instance output by our reduction. The set of indeterminants x of the (IP) instance is $\{x(\{a, b\}, c_H(a), c_H(b))\colon \{a, b\} \in E(H)\}$. Notice that for any $\{a, b\} \in E(H)$, there are two indeterminants $x(\{a, b\}, c_H(a), c_H(b))$ and $x(\{a, b\}, c_H(b), c_H(a))$ associated with it. Thus the cardinality of x is upper bounded by $2|E(H)| = 2m$. Recall that $\{v_1, \ldots, v_\ell\} = V(G)$ and $c_G(v_i) = i$ for all $i \in [\ell]$, where $\ell = |V(G)|$. For each $v_i \in V(G)$ we define $2d_G(v_i) - 1$ many constraints as explained below. Let $r = d_G(v_i)$ and $N_G(v_i) = \{v_{j_1}, \ldots, v_{j_r}\}$. The constraints for $v_i \in V(G)$ are the following. For all $q \in [r]$,

$$\sum_{e \in E_H(i, j_q)} x(e, i, j_q) = 1 \tag{1}$$

The constraints of the form above encode the "selection" constraint in Partitioned Subgraph Isomorphism, which says that for every edge $\{i, j\}$ in G, we must pick an edge in H which has one endpoint in color class i and the other in color class j. For all $q \in [r-1]$,

$$\sum_{\substack{\{a,b\} \in E_H(i, j_q) \\ a \in c_H^{-1}(i)}} a \cdot x(\{a, b\}, i, j_q) + \sum_{\substack{\{a,b'\} \in E_H(i, j_{q+1}) \\ a \in c_H^{-1}(i)}} (n - a) \cdot x(\{a, b'\}, i, j_{q+1}) = n \tag{2}$$

For each $\{v_i, v_j\} \in E(G)$ with $i < j$, we define the following constraint in the (IP) instance.

$$\sum_{\substack{\{a,b\} \in E_H(i, j) \\ a \in c_H^{-1}(i)}} \alpha(\{a, b\}) \cdot x(\{a, b\}, i, j) + \sum_{\substack{\{a,b\} \in E_H(i, j) \\ b \in c_H^{-1}(j)}} (m - \alpha(\{a, b\})) \cdot x(\{a, b\}, j, i) = m \tag{3}$$

The two sets of constraints above enforce the property that for any color class i in H, the set of edges that we have selected in the solution among those with exactly one endpoint in i, in fact have the same endpoint in the color class i. Together these constraints allow one to reconstruct the solution to the Partitioned Subgraph Isomorphism instance from a feasible solution for the resulting (IP) instance. Clearly, the number of rows in A is $|E(G)| + \sum_{v \in V(G)} 2d_G(v) - 1 \leq 5k$ and number of columns in A is $2m$. In order to prove Theorem 4, we first show that (G, H, c_G, c_H) is a Yes instance of Partitioned Subgraph Isomorphism if and only if $Ax = b, x \geq 0$ is feasible and if $Ax = b, x \geq 0$ is feasible, then for any solution x^*, each entry of x^* belongs to $\{0, 1\}$.

4 Path-width parameterization: SETH bounds

We prove Theorems 7 and 8 by giving reductions from CNF-SAT. At this point, one might be tempted to start the reduction from k-CNF SAT as seen in [2]. However, the fact that in our case we also need to control the path-width of the reduced instance poses serious technical difficulties if one were to take this route. Therefore, we take a different route and reduce from CNF-SAT which allows us to construct appropriate gadgets for propagation of consistency in our instance while simultaneously controlling the path-width. Moreover, the parameters in the reduced instances are required to obey certain strict conditions. For example, the reduction we give to prove Theorem 7 must output an instance of (IP), where the path-width of the column matroid $M(A)$ of the constraint matrix A is a constant and the upper bound on the largest entry in b depends on the path-width. Similarly, in the reduction used to prove Theorem 8, we need to construct an instance of (IP) where the largest entry in the target vector is upper bounded by a constant. These stringent requirements on the parameters make the SETH-based reductions quite challenging. However, reductions under SETH are allowed to take super polynomial time – they can even take $2^{(1-\epsilon)n}$ time for some $\epsilon > 0$, where n is the number of variables in the instance of CNF-SAT. This freedom to avail exponential time in SETH-based reductions is used crucially in the proofs of Theorems 7 and 8.

Now we give an overview of the reduction used to prove Theorem 7. Let ψ be an instance of CNF-SAT with n variables and m clauses. Given ψ and a fixed constant $c \geq 2$, we construct an instance $A_{(\psi,c)}x = b_{(\psi,c)}, x \geq 0$ of (IP) satisfying certain properties. Since for every $c \geq 2$, we have a different $A_{(\psi,c)}$ and $b_{(\psi,c)}$, this can be viewed as a family of instances of (IP). In particular our main technical lemma is the following.

▶ **Lemma 11.** *Let ψ be an instance of* CNF-SAT *with n variables and m clauses. Let $c \geq 2$ be a fixed integer. Then, in time $\mathcal{O}(m^2 2^{\frac{n}{c}})$, we can construct an instance $A_{(\psi,c)}x = b_{(\psi,c)}, x \geq 0$, of (IP) with the following properties.*

(a.) *ψ is satisfiable if and only if $A_{(\psi,c)}x = b_{(\psi,c)}, x \geq 0$ is feasible.*
(b.) *The matrix $A_{(\psi,c)}$ is non-negative and has dimension $\mathcal{O}(m) \times \mathcal{O}(m2^{\frac{n}{c}})$.*
(c.) *The path-width of the column matroid of $A_{(\psi,c)}$ is at most $c + 4$.*
(d.) *The largest entry in $b_{(\psi,c)}$ is at most $2^{\lceil \frac{n}{c} \rceil} - 1$.*

Once we have Lemma 11, we prove Theorem 7 using the fact that if we have an algorithm $\mathcal{A}$ solving (IP) in time $f(k)(\|b\|_\infty + 1)^{(1-\epsilon)k}(mn)^a$ for some $\epsilon, a > 0$, then we can use this algorithm to refute SETH. In particular, given an instance ψ of CNF-SAT, we choose an appropriate c depending only on ϵ and a, construct an instance $A_{(\psi,c)}x = b_{(\psi,c)}, x \geq 0$, of (IP), and run $\mathcal{A}$ on it. Our careful choice of c will imply a faster algorithm for CNF-SAT, refuting SETH. More formally, we choose c to be an integer such that $(1-\epsilon) + \frac{4(1-\epsilon)}{c} + \frac{a}{c} < 1$. Then the total running time to test whether ψ is satisfiable, is the time require to construct $A_{(\psi,c)}x = b_{(\psi,c)}, x \geq 0$ plus the time required by $\mathcal{A}$ to solve the constructed instance of (IP). That is, the time required to test whether ψ is satisfiable is

$$\mathcal{O}(m^2 2^{\frac{n}{c}}) + f(c+4)2^{\frac{n}{c}(1-\epsilon)(c+4)}2^{\frac{a \cdot n}{c}}m^{\mathcal{O}(1)} = 2^{\left((1-\epsilon)+\frac{4(1-\epsilon)}{c}+\frac{a}{c}\right)n}m^{\mathcal{O}(1)} = 2^{\epsilon' n}m^{\mathcal{O}(1)},$$

where $\epsilon' < 1$ is a constant depending on the choice of c. It is important to note that the utility of the reduction described in Lemma 11 is extremely sensitive to the value of the numerical parameters involved. In particular, even when the path-width blows up slightly, say up to δc, or when the largest entry in $b_{(\psi,c)}$ blows up slightly, say up to $2^{\delta \frac{n}{c}}$, for some $\delta > 1$, then the calculation above will *not* give us the desired refutation of SETH. Thus, the challenging part of the reduction described in Lemma 11 is making it work under these strict restrictions on the relevant parameters and we focus on this part in the extended abstract.

As stated in Lemma 11, in our reduction, we need to obtain a constraint matrix with small path-width. An important first step towards this is understanding what a matrix of small path-width looks like. We first give an intuitive description of the structure of such matrices. Let A be a $m \times n$ matrix of small path-width and let $M(A)$ be the column matroid of A. For any $i \in \{1, \ldots, n-1\}$, let $A|\{1, \ldots i\}$ denote the set of columns (or vectors) in A whose index is at most i (that is, the first i columns) and let $A|\{i+1, \ldots n\}$ denote the set of columns with index strictly greater than i. The path-width of $M(A)$ is at most $\max_i \dim\langle \text{span}(A|\{1, \ldots, i\}) \cap \text{span}(A|\{i+1, \ldots, n\})\rangle + 1$. Consequently, in order to obtain a bound on the pathwidth, it is sufficient to bound $\dim\langle \text{span}(A|\{1, \ldots, i\}) \cap \text{span}(A|\{i + 1, \ldots, n\})\rangle$ for every $i \in [n]$.

The construction used in Lemma 11 takes as input an instance ψ of CNF-SAT with n variables and a fixed integer $c \geq 2$, and outputs an instance $A_{(\psi,c)}x = b_{(\psi,c)}, x \geq 0$, of (IP), that satisfies all four properties of the lemma. Let X denote the set of variables in the input CNF-formula $\psi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$. For the purposes of the present discussion we assume that c divides n. We partition the variable set X into c blocks $X_0, \ldots, X_{c-1}$, each of size $\frac{n}{c}$. Let $\mathcal{X}_i$, $i \in \{0, \ldots, c-1\}$, denote the set of assignments of variables corresponding to X_i. Set $\ell = \frac{n}{c}$ and $L = 2^\ell$. Clearly, the size of $\mathcal{X}_i$ is upper bounded by $2^{\frac{n}{c}} = 2^\ell = L$. We denote the assignments in $\mathcal{X}_i$ by $\phi_0(X_i), \phi_1(X_i), \ldots, \phi_{L-1}(X_i)$. To construct the matrix $A_{(\psi,c)}$, we view "each of these assignments as a different assignment for each clause". In other words we have separate sets of variables in the constraints corresponding to different pairs (C_r, X_i), where C_r is a clause and X_i is a block in the partition of X. That is for each clause C_r and block X_i, we have variables $\{y_{C_r,i,a}\ a \in \mathbb{Z}_{2L}\ \}$. In other words for each C_r and assignment $\phi_a(X_i)$, $a \in \mathbb{Z}_L$, we have two variables $y_{C_r,i,2a}$ and $y_{C_r,i,2a+1}$. For any clause C_r, $i \in \mathbb{Z}_c$ and $a \in \mathbb{Z}_{2L}$, assigning value 1 to $y_{C,i,a}$ corresponds to choosing an assignment $\phi_{\lfloor \frac{a}{2} \rfloor}(X_i)$ for X_i. In our reduction we will create the following set of constraints.

$$\sum_{\substack{i \in [c], a \in \mathbb{Z}_{2L} \text{ such that} \\ a \text{ is even and} \\ \phi_{\lfloor \frac{a}{2} \rfloor}(X_i) \text{ satisfies } C}} y_{C,i,a} \quad = \quad 1 \qquad \text{for all } C \in \mathcal{C} \tag{4}$$

$$\sum_{a \in \mathbb{Z}_{2L}} y_{C,i,a} \quad = \quad 1 \qquad \text{for all } C \in \mathcal{C} \text{ and } i \in \mathbb{Z}_c \tag{5}$$

Equation (4) takes care of satisfiability of clauses, while Equation (5) allows us to pick only one assignment from $\{\phi_0(X_i), \phi_1(X_i), \ldots, \phi_{L-1}(X_i)\}$ per clause C and block X_i. Note that this implies that we will choose an assignment in $\mathcal{X}_i$ for each clause C_r. That way we might choose m assignments from $\mathcal{X}_i$ corresponding to m different clauses. However, for the backward direction of the proof, it is important that we choose the *same* assignment from $\mathcal{X}_i$ for each clause. This will ensure that we have selected an assignment to the variables in X_i. Towards this we will have a third set of constraints as follows.

$$\sum_{a \in \mathbb{Z}_{2L}} \left(\lfloor \frac{a}{2} \rfloor \cdot y_{C_r,i,a} \right) + \left((L - 1 - \lfloor \frac{a}{2} \rfloor) y_{C_{r+1},i,a} \right) = L - 1 \ \forall r \in [m-1]\ ,\ i \in \mathbb{Z}_c \tag{6}$$

Equation (6) enforce consistencies of assignments of blocks across clauses in a *sequential* manner. That is, for any block X_i, we make sure that the two variables set to 1 corresponding to (C_r, X_i) and (C_{r+1}, X_i) are consistent for any $r \in \{1, \ldots, m-1\}$, as opposed to checking the consistency for every pair (C_r, X_i) and $(C_{r'}, X_i)$ for $r \neq r'$. Thus in some sense these consistencies *propagate*. Furthermore, the idea of making consistency in a sequential manner also allows us to bound the path-width of column matroid of $A_{(\psi,c)}$ by $c+4$.

The proof technique for Theorem 8 is similar to that for Theorem 7. This is achieved by modifying the matrix $A_{(\psi,c)}$ constructed in the reduction described for Lemma 11. The largest entry in $A_{(\psi,c)}$ is $2^{\frac{n}{c}}-1$ (see Equation (6)). So each of these values can be represented by a binary string of length at most $\ell = \frac{n}{c}$. We remove each row, say row indexed by γ, with entries greater than 1 and replace it with $\frac{n}{c}$ rows, $\gamma_1, \ldots, \gamma_\ell$. Where, for any j, if the value $A_{(\psi,c)}[\gamma, j] = W$ then $A_{(\psi,c)}[\gamma_k, j] = \eta_k$, where η_k is the k^{th} bit in the ℓ-sized binary representation of W. This modification reduces the largest entry in $A_{(\psi,c)}$ to 1 and increases the path-width from constant to approximately n. Finally, we set all the entries in $b_{(\psi,c)}$ to be 1. This concludes the overview of our reductions.

4.1 Proof of Theorem 7

In this section we give a more detailed sketch of the proof of Theorem 7. Towards this, we first present the main details in the proof of our most technical lemma (Lemma 11).

Let $\psi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be an instance of CNF-SAT with variable set $X = \{x_1, x_2, \ldots, x_n\}$ and let $c \geq 2$ be a fixed constant given in the statement of Lemma 11. We construct the instance $A_{(\psi,c)}x = b_{(\psi,c)}, x \geq 0$ of (IP) as follows.

Construction. Let $\mathcal{C} = \{C_1, \ldots, C_m\}$. Without loss of generality, we assume that n is divisible by c, otherwise we add at most c dummy variables to X such that $|X|$ is divisible by c. We divide X into c blocks $X_0, X_1, \ldots, X_{c-1}$. That is $X_i = \{x_{\frac{i\cdot n}{c}+1}, x_{\frac{i\cdot n}{c}+2}, \ldots, x_{\frac{(i+1)\cdot n}{c}}\}$ for each $i \in \mathbb{Z}_c$. Let $\ell = \frac{n}{c}$ and $L = 2^\ell$. For each block X_i, there are exactly 2^ℓ assignments. We denote these assignments by $\phi_0(X_i), \phi_1(X_i), \ldots, \phi_{L-1}(X_i)$.

Now, we create $m \cdot c \cdot 2^{\ell+1}$ variables; they are named $y_{C,i,a}$, where $C \in \mathcal{C}$, $i \in \mathbb{Z}_c$ and $a \in \mathbb{Z}_{2L} = \mathbb{Z}_{2^{\ell+1}}$. In other words, for a clause C, a block X_i and an assignment $\phi_a(X_i)$, we create two variables; they are $y_{C,i,2a}$ and $y_{C,i,2a+1}$. Then, we create the (IP) constraints given by Equations (4), (5), and (6).

This completes the construction of (IP) instance. Let $A_{(\psi,c)}y = b_{(\psi,c)}$ be the (IP) instance defined using Equations (4), (5), and (6). The purpose of Equation (4) is to ensure satisfiability of all the clauses. Because of Equation (5), for each clause C and for each block X_i, we select only one assignment. Notice, that, so far it is allowed to choose many assignments from a block X_i, for different clauses. To ensure the consistency of assignments in each block across clauses, we added a system of constraints (Equation (6)). Equation (6) ensures the consistency of assignments in the adjacent clauses (in the order $C_1, \ldots, C_m$). Thus, the consistency of assignments propagates in a sequential manner. Notice that number constraints defined by Equations (4), (5), and (6) are m, $m \cdot c$ and $(m-1)\cdot c$, respectively. The number of variables is $m \cdot c \cdot 2^{\ell+1}$. Also notice that all the coefficients in Equations (4), (5) and (6) are non-negative. This implies that $A_{(\psi,c)}$ is non-negative and has dimension $\mathcal{O}(m) \times \mathcal{O}(m2^{\frac{n}{c}})$. Thus, the property $(b.)$ of Lemma 11 is satisfied. The largest entry in $b_{(\psi,c)}$ is $L-1 = 2^{\lceil \frac{n}{c} \rceil}-1$ (see Equation (6)) and hence the property $(d.)$ of Lemma 11 is satisfied. The complete details for the proof of property $(a.)$ can be found in the appended full version. Moving forward, we simplify the notation by using A instead of $A_{(\psi,c)}$ and b instead of $b_{(\psi,c)}$.

Now we need to prove property $(c.)$ of Lemma 11. That is the path-width of A is at most $c+4$. Towards that we need to understand the structure of matrix A. We decompose the matrix A into m disjoint submatrices $B_1, \ldots B_m$ which are disjoint and cover all the non-zero entries in the matrix A. First we define some notations and fix the column indices of A corresponding the the variables in the constraints. Let Y denote the set $\{y_{C,i,a} \mid C \in \mathcal{C}, i \in \mathbb{Z}_c, a \in \mathbb{Z}_{2L}\}$ of variables in the constraints defined by Equations (4), (5) and (6). These variables can be partitioned into $\biguplus_{C\in\mathcal{C}} Y_C$, where $Y_C = \{y_{C,i,a} \mid i \in \mathbb{Z}_c, a \in \mathbb{Z}_{2L}\}$. Further for each $C \in \mathcal{C}$, Y_C can be partitioned into $\bigcup_{i\in\mathbb{Z}_c} Y_{C,i}$, where $Y_{C,i} = \{y_{C,i,a} \mid a \in \mathbb{Z}_{2L}\}$. The set of columns indexed by $[r \cdot c\dot{2}^{\ell+1}] \setminus [(r-1)\cdot c \cdot 2^{\ell+1}]$, for any $r \in [m]$, corresponds to the set of variables in Y_{C_r}. Among the set of columns corresponding to Y_C, the first $2^{\ell+1}$ columns corresponds to the variables in $Y_{C,1}$, second $2^{\ell+1}$ columns corresponds to the variables in $Y_{C,2}$, and so on. Among the set of columns corresponds to $Y_{C,i}$ for any $C \in \mathcal{C}$ and $i \in \mathbb{Z}_c$, the first two columns corresponds to the variable $y_{C,i,0}$ and $y_{C,i,1}$, and second two columns corresponds to the variables $y_{C,i,2}$ and $y_{C,i,3}$, and so on.

Now we move to the description of B_j, $j \in [m]$. The matrix B_j will cover the coefficients of Y_{C_j} in Equations (4), (5) and (6). In other words B_j covers the non-zero entries in the columns corresponding to Y_{C_j}, i.e, in the columns of A indexed by $[j \cdot c \cdot 2^{\ell+1}] \setminus [(j-1)\cdot c \cdot 2^{\ell+1}]$. Now we explain these submatrices. Each matrix B_j has $c \cdot 2^{\ell+1}$ columns; each of them corresponds to a variable in Y_{C_j}. Each row in A corresponds to a constraint in the system of equations defined by Equations (4), (5) and (6). So we use notations $f(C_1), \dots f(C_m)$ to represents the constraints defined by Equations (4). Similarly we use notations $\{s(C,i) \mid C \in \mathcal{C}, i \in \mathbb{Z}_c\}$ and $\{t(C,i) \mid C \in \mathcal{C}, i \in \mathbb{Z}_c\}$ to represent the constraints defined by Equations (5) and (6), respectively.

Matrices B_r for $1 < r < m$. Matrix B_r is of dimension $(3c+1) \times (c \cdot 2^{\ell+1})$. The first c rows are defined by Equation (6). For $j \in [c]$, in i^{th} row, we have coefficients of Y_{C_r} from $t(C_{r-1}, i)$. In the $(c+1)^{st}$ row of B_r, we have coefficients of Y_{C_r} from $f(C_r)$. For $i \in [c]$, the rows indexed by $c+1+i$ and $2c+1+i$ are defined as follows. In the $(c+1+i)^{th}$ row of B_r, we have coefficients of Y_{C_r} from $s(C_r, i)$ while in the $(2c+1+i)^{th}$ row of B_r, we have coefficients of Y_{C_r} from $t(C_r, i)$. This completes the definition of B_r. By their role in the reduction, the matrix B_r is partitioned in to four parts. The part composed of the first c rows is called the *predecessor matching part.* The part composed of the row indexed by $c+1$ is called the *evaluation part* of B_1. The part composed of rows indexed by $c+2, c+3, \dots, 2c+1$ is called *selection part* and the part composed of last c rows is called *successor matching part.* That is the entries of B_1 are as follows, where $i \in \mathbb{Z}_c$ and $a \in \mathbb{Z}_L$.

The predecessor matching part is defined by

$$B_r[i+1, i\cdot 2^{\ell+1} + 2a + 1] = B_r[i+1, i\cdot 2^{\ell+1} + 2a + 2] = L - 1 - a. \tag{7}$$

The evaluation part is defined by

$$B_r[c+1, i\cdot 2^{\ell+1} + 2a + 2] = 0, \tag{8}$$

and

$$B_r[c+1, i\cdot 2^{\ell+1} + 2a + 1] = \begin{cases} 1, & \text{if } \phi_a(X_i) \text{ satisfies } C_r, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

The selection part for B_r is defined as

$$B_r[c+2+i, i\cdot 2^{\ell+1} + 2a + 1] = B_r[c+2+i, i\cdot 2^{\ell+1} + 2a + 2] = 1, \tag{10}$$

The successor matching part for B_r is defined as

$$B_r[2c+2+i, i\cdot 2^{\ell+1} + 2a + 1] = B_r[2c+2+i, i\cdot 2^{\ell+1} + 2a + 2] = j. \tag{11}$$

All other entries in B_r, which are not listed above, are zero. That is, for all $i, i' \in \mathbb{Z}_c$ and $g \in [2^{\ell+1}]$ such that $i \neq i'$,

$$B_r[i+1, i' \cdot 2^{\ell+1} + g] = 0, \tag{12}$$
$$B_r[c+2+i, i' \cdot 2^{\ell+1} + g] = 0, \text{ and} \tag{13}$$
$$B_r[2c+2+i, i' \cdot 2^{\ell+1} + g] = 0. \tag{14}$$

Matrices B_1 and B_m. These have a slightly different structure. Informally, B_1 and B_m can be defined like B_r, $1 < r < m$, but we delete first c rows to get B_1 and delete last c rows to get B_m. A brief description of B_1 and B_m is given below.

Matrix B_1 is of dimension $(2c+1) \times (c \cdot 2^{\ell+1})$. In the first row of B_1, we have coefficients of Y_{C_1} from $f(C_1)$. For $i \in \mathbb{Z}_c$, the rows indexed by $2+i$ and $c+2+i$ are defined as follows. In the $(2+i)^{th}$ row of B_1, we have coefficients of Y_{C_1} from $s(C_1, i)$ while in the $(c+2+i)^{th}$ row of B_1, we have coefficients of Y_{C_1} from $t(C_1, i)$.

Matrix B_m is of dimension $(2c+1) \times (c \cdot 2^{\ell+1})$. For $j \in [c]$, in i^{the} row, we have coefficients of Y_{C_m} from $t(C_{m-1}, i)$. In the $(c+1)^{st}$ row of B_r, we have coefficients of Y_{C_m} from $f(C_m)$. In the $(c+1+i)^{th}$ row of B_m, we have coefficients of Y_r from $s(C_m, i)$.

Matrix A. Now we explain how the matrix A is formed from $B_1, \ldots, B_m$. The matrices $B_1, \ldots, B_m$ are disjoint submatrices of A and they cover all non zero entries of A. Informally, the submatrices $B_1, \ldots, B_m$ form a chain such that the rows corresponding to the successor matching part of B_r will be the same as the rows in the predecessor matching part of B_{r+1} (because of Equation (6)). Formally, let $I_1 = [2c+1]$ and $I_m = [(m-1)(2c+1) + (c+1)] \setminus [(m-1)(2c+1) - c]$. For every $1 < r < m$, let $I_r = [r(2c+1)] \setminus [(r-1)(2c+1) - c]$, and for $r \in [m]$, let $J_r = [r \cdot c \cdot 2^{\ell+1}] \setminus [(r-1) \cdot c \cdot 2^{\ell+1}]$. Now for each $r \in [m]$, the matrix $A[I_r, J_r] := B_r$. All other entries of A not belonging to any of the submatrices $A[I_r, J_r]$ are zero.

Towards upper bounding the path-width of A, we start with some notations. We partition the set of columns of A into m parts $J_1, \ldots, J_m$ (we have already defined these sets) with one part per clause. For each $r \in [m]$, J_r is the set of columns associated with Y_{C_r}. We further divide J_r into c equal parts, one per variable set $Y_{C_r,i}$. These parts are

$$P_{r,i} = \{(r-1)c \cdot 2^{\ell+1} + i \cdot 2^{\ell+1} + 1, \ldots, (r-1)c \cdot 2^{\ell+1} + (i+1) \cdot 2^{\ell+1}\},\ i \in \mathbb{Z}_c.$$

In other words, $P_{r,i}$ is the set of columns corresponding to $Y_{C_r,i}$ and $|P_{r,i}| = 2^{\ell+1}$. We also put $n' = m \cdot c \cdot 2^{\ell+1}$ to be the number of columns in A.

▶ **Lemma 12.** *The path-width of the column matroid of A is at most $c+4$*

Proof. Recall that $n' = m \cdot c \cdot 2^{\ell+1}$, is the number of columns in A and m' the number of rows in A. To prove that the path-width of A is $\leq c+4$, it suffices to show that for all $j \in [n'-1]$,

$$\dim\langle \text{span}(A|\{1, \ldots, j\}) \cap \text{span}(A|\{j+1, \ldots, n'\})\rangle \leq c+3. \tag{15}$$

The idea for proving Equation (15) is based on the following observation. For $V' = A|\{1, \ldots, j\}$ and $V'' = A|\{j+1, \ldots, n'\}$, let $I = \{q \in [m'] \mid \text{there exist } v' \in V' \text{ and } v'' \in V'' \text{ such that } v'[q] \neq v''[q] \neq 0\}$. Then the dimension of $\text{span}(V') \cap \text{span}(V'')$ is at most $|I|$. Thus to prove (15), for each $j \in [n'-1]$, we construct the corresponding set I and show that its cardinality is at most $c+3$.

We proceed with the details. Let $v_1, v_2, \ldots, v_{n'}$ be the column vectors of A. Let $j \in [n'-1]$. Let $V_1 = \{v_1, \ldots, v_j\}$ and $V_2 = \{v_{j+1}, \ldots, v_{n'}\}$. We need to show that $\dim\langle \text{span}(V_1) \cap$

$\text{span}(V_2)\rangle \leq c+3$. Let $I' = \{q \in [m'] \mid$ there exists $v \in V_1$ and $v' \in V_2$ such that $v[q] \neq 0 \neq v'[q]\}$. We know that $[n']$ is partitioned into parts $P_{r',i'}, r' \in [m], i' \in \mathbb{Z}_c$. We fix $r \in [m]$ and $i \in \mathbb{Z}_c$ such that $j \in P_{r,i}$.

Let $j = (r-1)c \cdot 2^{\ell+1} + i \cdot 2^{\ell+1} + g$, where $g \in [2^{\ell+1}]$. Let $q_1 = \max\{0, (r-1)(2c+1) - c\}$, $q_2 = r(2c+1)$, $j_1 = (r-1) \cdot c \cdot 2^{\ell+1}$, and $j_2 = r \cdot c \cdot 2^{\ell+1}$. Then $[q_2] \setminus [q_1] = I_r$ and $[j_2] \setminus [j_1] = J_r$ (recall the definition of sets I_r and J_r).

By the decomposition of matrix A, for every $q > q_2$ and for every vector $v \in V_1$, we have $v[q] = 0$. Also, for every $q \leq q_1$ and for any $v \in V_2$, we have that $v[q] = 0$. This implies that $I' \subseteq [q_2] \setminus [q_1] = I_r$. Now we partition I_r into 4 parts: R_1, R, S, and R_2, These parts are defined as follows.

$$\begin{aligned} R_1 &= \begin{cases} \emptyset, & \text{if } r = 1, \\ \{(r-2)(2c+1) + i' \mid i' \in \mathbb{Z}_c\}, & \text{otherwise,} \end{cases} \\ R &= \{(r-1)(2c+1)+1\}, \\ S &= \{(r-1)(2c+1) + 2 + i' \mid i' \in \mathbb{Z}_c]\}, \\ R_2 &= \begin{cases} \emptyset, & \text{if } r = m, \\ \{(r-1)(2c+1) + c + 2 + i' \mid i' \in \mathbb{Z}_c\}, & \text{otherwise} \end{cases} \end{aligned} \tag{16}$$

We complete the proof of the lemma by proving the following series of claims. We first show that for each $r' \in [m]$ such that $q \notin I_{r'}$ and $j'' \in J_{r'}$, $v_{j''}[q] = 0$. Following that, we show that $|I' \cap R_1| \leq c - (i-1)$. The final two claims in this series of claims are (i) $|I' \cap R_2| \leq i$, and (ii) $|I' \cap S| \leq 1$.

With the help of these claims, we can conclude the following. $|I'| = |I' \cap I_r|$ (since $I' \subseteq I_r$) and $|I' \cap I_r| = |I' \cap R_1| + |I' \cap R| + |I' \cap S| + |I' \cap R_2|$ (by (16)), which implies that $|I'| \leq c - (i-1) + 1 + 1 + i = c + 3$. This completes the proof of the lemma. ◀

5 Conclusion

While Theorems 3 and 4 come close to the bound of Proposition 1, the precise multivariate complexity of (IP) with respect to the parameters n, m, Δ, and $\|b\|_\infty$ is not fully clear and our work leaves some unanswered questions regarding the landscape of tradeoffs between the parameters. For instance, is it possible to solve (IP) in time $(m \cdot n \cdot \Delta)^{o(m)} \cdot (\|b\|_\infty)^{\mathcal{O}(1)}$, or $(m \cdot n \cdot \Delta \cdot \|b\|_\infty)^{o(m)}$? Or could one improve our lower bound results to rule out such algorithms? While our SETH-based lower bounds for (IP) with non-negative constraint matrix are tight for path-width parameterization, there is a "$(\|b\|_\infty + 1)^k$ to $(\|b\|_\infty + 1)^{2k}$ gap" between lower and upper bounds for branch-width parameterization. Closing this gap is a natural question.

The bottleneck in the algorithm of Cunningham and Geelen is the following subproblem. We are given two vector sets A and B of partial solutions, each set of size at most $(\|b\|_\infty + 1)^k$. We need to construct a new vector set C of partial solutions, where the set C will have size at most $(\|b\|_\infty + 1)^k$ and each vector from C is the *sum* of a vector from A and a vector from B. Thus to construct the new set of vectors, one has to go through all possible pairs of vectors from both sets A and B, which takes time roughly $(\|b\|_\infty + 1)^{2k}$.

A tempting approach towards speeding up this particular step could be the use of *fast subset convolution* or *matrix multiplication* tricks, which work very well for "join" operations in dynamic programming algorithms over tree and branch decompositions of graphs [5, 18, 4], see also [3, Chapter 11]. Unfortunately, we have reason to suspect that these tricks may *not* help for matrices: solving the above subproblem in time $(\|b\|_\infty + 1)^{(1-\epsilon)2k} n^{\mathcal{O}(1)}$ for any $\epsilon > 0$ would imply that 3-SUM is solvable in time $n^{2-\epsilon}$, which is believed to be unlikely.

References

1 William H. Cunningham and Jim Geelen. On integer programming and the branch-width of the constraint matrix. In *Proceedings of the 12th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, volume 4513 of *Lecture Notes in Comput. Sci.*, pages 158–166. Springer, 2007.

2 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. In *Proceedings of the 27th IEEE Conference on Computational Complexity (CCC)*, pages 74–84. IEEE, 2012. doi:10.1109/CCC.2012.36.

3 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

4 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 150–159. IEEE, 2011.

5 Frederic Dorn. Dynamic programming and fast matrix multiplication. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, volume 4168 of *Lecture Notes in Comput. Sci.*, pages 280–291. Springer, Berlin, 2006.

6 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the steinitz lemma. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 808–816. SIAM, 2018.

7 G. B. Horn and Frank R. Kschischang. On the intractability of permuting a block code to minimize trellis complexity. *IEEE Trans. Information Theory*, 42(6):2042–2048, 1996. doi:10.1109/18.556701.

8 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

9 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity. *J. Computer and System Sciences*, 63(4):512–530, 2001.

10 K. Jansen and L. Rohwedder. On Integer Programming and Convolution. *ArXiv e-prints*, 2018. arXiv:1803.04744.

11 Jisu Jeong, Eun Jung Kim, and Sang-il Oum. Constructive algorithm for path-width of matroids. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1695–1704. SIAM, 2016. doi:10.1137/1.9781611974331.ch116.

12 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of operations research*, 12(3):415–440, 1987.

13 Hendrik W Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of operations research*, 8(4):538–548, 1983.

14 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 777–789. SIAM, 2011.

15 Dániel Marx. Can you beat treewidth? *Theory of Computing*, 6(1):85–112, 2010. arXiv:toc:v006/a005.

16 Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981. doi:10.1145/322276.322287.

17 Neil Robertson and Paul D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Combinatorial Theory Ser. B*, 52(2):153–190, 1991. doi:10.1016/0095-8956(91)90061-N.

18 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA)*, volume 5757 of *Lecture Notes in Comput. Sci.*, pages 566–577. Springer, 2009.

Symmetry Exploitation for Online Machine Covering with Bounded Migration

Waldo Gálvez
IDSIA, USI-SUPSI
Lugano, Switzerland
waldo@idsia.ch

José A. Soto
Departamento de Ingeniería Matemática & CMM, Universidad de Chile
Santiago, Chile
jsoto@dim.uchile.cl

José Verschae
Facultad de Matemáticas & Escuela de Ingeniería, Pontificia Universidad Católica de Chile
Santiago, Chile
jverschae@uc.cl

Abstract

Online models that allow recourse are highly effective in situations where classical models are too pessimistic. One such problem is the online machine covering problem on identical machines. In this setting, jobs arrive one by one and must be assigned to machines with the objective of maximizing the minimum machine load. When a job arrives, we are allowed to reassign some jobs as long as their total size is (at most) proportional to the processing time of the arriving job. The proportionality constant is called the *migration factor* of the algorithm.

By rounding the processing times, which yields useful structural properties for online packing and covering problems, we design first a simple $(1.7+\varepsilon)$-competitive algorithm using a migration factor of $O(1/\varepsilon)$ which maintains at every arrival a locally optimal solution with respect to the Jump neighborhood. After that, we present as our main contribution a more involved $(4/3+\varepsilon)$-competitive algorithm using a migration factor of $\tilde{O}(1/\varepsilon^3)$. At every arrival, we run an adaptation of the *Largest Processing Time first* (LPT) algorithm. Since the new job can cause a complete change of the assignment of smaller jobs in both cases, a low migration factor is achieved by carefully exploiting the highly symmetric structure obtained by the rounding procedure.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms, Theory of computation → Online algorithms

Keywords and phrases Machine Covering, Bounded Migration, Online, Scheduling, LPT

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.32

Related Version A full version of the paper is available at [9], `https://arxiv.org/abs/1612.01829`.

Funding This work was partially supported by SNSF Grant APXNET 200021_159697/1 and CONICYT-Chile through projects FONDECYT 1181527 and 1181180, PCI PII 20150140 and PIA AFB170001.

© Waldo Gálvez, José A. Soto, and José Verschae;
licensed under Creative Commons License CC-BY

1 Introduction

We consider a fundamental load balancing problem where n jobs need to be assigned to m identical parallel machines. Each job j is fully characterized by a non-negative processing time p_j. Given an assignment of jobs, the load of a machine is the sum of the processing times of jobs assigned to it. The *machine covering problem* asks for an assignment of jobs to machines maximizing the load of the least loaded machine.

This problem is well known to be strongly NP-hard and allows for a polynomial-time approximation scheme (PTAS) [21]. A well studied algorithm for this problem is the *Largest Processing Time First* rule (LPT), that sorts the jobs non-increasingly and assigns them iteratively to the least loaded machine. Deuermeyer et al. [5] show that LPT is a $\frac{4}{3}$-approximation and that this factor is asymptotically tight; later, Csirik et al. [4] refine the analysis giving a tight bound for each m.

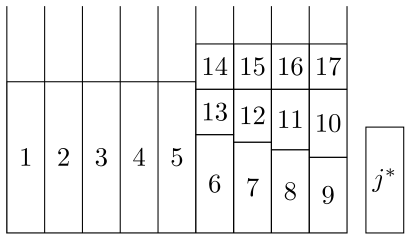
In the online setting jobs arrive one after another, and at the moment of an arrival, we must decide on a machine to assign the arriving job. This natural problem does not admit a constant competitive ratio. Deterministically, the best possible competitive ratio is m [21], while randomization allows for a $\tilde{O}(\sqrt{m})$-competitive algorithm, which is the best possible up to logarithmic factors [1].

Dynamic model. These negative facts motivate the study of a relaxed online scenario with *bounded migration*. Unlike the classic online model, when a new job j arrives we are allowed to reassign other jobs. More precisely, given a constant $\beta > 0$, we can migrate jobs whose total size is upper bounded by βp_j. The value β is called the *migration factor* and it accounts for the robustness of the computed solutions. In one extreme, we can model the usual online framework by setting $\beta = 0$. In the other extreme, setting $\beta = \infty$ allows to compute the optimal offline solution in each iteration. Our main interest is to understand the exact trade-off between the migration factor β and the competitiveness of our algorithms. Besides being a natural problem with an interesting theoretical motivation, its original purpose was to find good algorithms for a problem in the context of Storage Area Networks (SAN) [17].

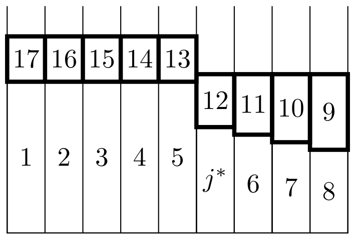
Local search and migration. The local search method has been extensively used to tackle different hard combinatorial problems, and it is closely related to online algorithms where recourse is allowed. This comes from the fact that simple local search neighborhoods allow to get considerably improved solutions while having accurate control over the recourse actions needed, and in some cases even a bounded number of local moves leads to substantially improved solutions (see [15, 10, 14] for examples in network design problems).

Related Work. Sanders et al. [17] develop online algorithms for load balancing problems with migration. For the makespan minimization objective, where the aim is to minimize the maximum load, they give a $(1+\varepsilon)$-competitive algorithm with $2^{\tilde{O}(1/\varepsilon)}$. A mayor open problem in this area is to determine whether a migration factor of poly$(1/\varepsilon)$ is achievable.

The landscape for the machine covering problem is somewhat different. Sanders et al. [17] give a 2-competitive algorithm with migration factor 1, and this is until now the best competitive ratio known for any algorithm with constant migration factor. On the negative side, Skutella and Verschae [19] show that it is not possible to maintain arbitrarily near optimal solutions using a constant migration factor, giving a lower bound of 20/19 for the best competitive ratio achievable in that case. The lower bound is based on an instance where arriving jobs are very small, not allowing to migrate other jobs. This motivated



(a) LPT for the original instance and arriving job j^*.



(b) LPT for the new instance. Thick items correspond to migrated jobs.

Figure 1 $\Omega(m)$ migration factor needed to maintain LPT at the arrival of j^*.

the study of an amortized version, called *reassignment cost model*, where they develop a $(1+\varepsilon)$-competitive algorithm using a constant reassignment factor. They also show that if all arriving jobs are larger than $\varepsilon \cdot \mathrm{OPT}$, then there is a $(1+\varepsilon)$-competitive algorithm with constant migration factor.

Similar migration models have been studied for other packing and covering problems. For example, Epstein & Levin [6] design a $(1+\varepsilon)$-competitive algorithm for the online bin packing problem using a migration factor of $2^{\tilde{O}(1/\varepsilon^2)}$, which was improved later by Jansen & Klein [12] to poly$(1/\varepsilon)$ migration factor, and then further refined by Berndt et al. [2]. Also, for makespan minimization with preemption and other objectives, Epstein & Levin [7] design a best-possible online algorithm using a migration factor of $\left(1-\frac{1}{m}\right)$.

Regarding local search applied to load balancing problems, many neighborhoods have been studied such as *Jump*, *Swap*, *Push* and *Lexicographical Jump* in the context of makespan minimization on related machines [18], makespan minimization on restricted related machines [16], and also multi-exchange neighborhoods for makespan minimization on identical parallel machines [8]. For the case of machine covering, Chen et al. [3] study the Jump neighborhood in a game-theoretical context, proving that every locally optimal solution is 1.7-approximate and that this factor is tight.

Our Contribution. Our main result is a $(4/3+\varepsilon)$-competitive algorithm using poly$(1/\varepsilon)$ migration factor. This is achieved by running a carefully crafted version of LPT at the arrival of each new job. We would like to stress that, even though LPT is a simple and well studied algorithm in the offline context, directly running this algorithm in each time step in the online context yields an unbounded migration factor; see Figure 1 for an illustrative example.

To overcome this barrier, we first adapt a less standard procedure to round processing times in the online framework. The rounding reduces the possible number of sizes of jobs larger than $\Omega(\varepsilon\mathrm{OPT})$ (where OPT is the offline optimum value) to $\tilde{O}(1/\varepsilon)$ many numbers, and furthermore these values are multiples of a common number $g \in \Theta(\varepsilon^2\mathrm{OPT})$. This implies that the number of possible loads for machines having only big jobs is constant since they are multiples of g as well. Unlike known techniques used in previous work that yield similar results (see e.g. [13]), our rounding is well suited for online algorithms and helps simplifying the analysis as it does not depend on OPT (which varies through iterations).

In order to show the usefulness of the rounding procedure, we first present a simple $(1.7+\varepsilon)$-competitive algorithm using a migration factor of $O(1/\varepsilon)$. This algorithm maintains through the arrival of new jobs a locally optimal solution with respect to Jump for large jobs and a greedy assignment for small jobs on top of that. Although for general instances this can induce a very large migration factor as discussed before, for rounded instances we can have a very accurate control on the jumps needed to reach a locally optimal solution by exploiting the fact that there are constant many possible processing times for large jobs.

In the second part of the paper we proceed with the analysis of our $(4/3+\varepsilon)$-competitive algorithm. Here we crucially make use of the properties obtained by the rounding procedure to create symmetries. After a new job arrival we re-run the LPT algorithm for the new instance. While assigning a job to a current least loaded machine, since there is a constant number of possible machine loads, there will usually be multiple least loaded machines to assign the job. All options lead to different (but symmetric) solutions in terms of job assignments, all having the same load vector and thus the same objective value. Broadly speaking, the algorithm will construct one of these symmetric schedules, trying to maintain as many machines with the same assignments as in the previous time step. The analysis of the algorithm will rely on monotonicity properties implied by LPT which, coupled with rounding, implies that for every job size the increase in the number of machines with different assignments (w.r.t the solution of the previous time step) is constant. This finally yields a migration factor that only grows polynomially in $1/\varepsilon$. Finally, we give a lower bound of $17/16$ for the best competitive ratio achievable by an algorithm with constant migration, improving the bound on [19].

Due to space constraints, we defer most of the proofs to the full version [9].

2 Preliminaries

Consider a set of n jobs $\mathcal{J}$ and a set of m machines $\mathcal{M}$. In our problem, a solution or schedule $\mathcal{S} : \mathcal{J} \to \mathcal{M}$ corresponds to an assignment of jobs to machines. The set of jobs assigned to a machine i is then $\mathcal{S}^{-1}(i) \subseteq J$. The load of machine i in $\mathcal{S}$ corresponds to $\ell_i(\mathcal{S}) = \sum_{j\in\mathcal{S}^{-1}(i)} p_j$. The minimum load is denoted by $\ell_{\min}(\mathcal{S}) = \min_{i\in\mathcal{M}} \ell_i(\mathcal{S})$, and a machine i is said to be *least loaded* in $\mathcal{S}$ if $\ell_i(\mathcal{S}) = \ell_{\min}(\mathcal{S})$.

For an algorithm $\mathcal{A}$ and an instance $(\mathcal{J}, \mathcal{M})$, we denote by $\mathcal{S}_{\mathcal{A}}(\mathcal{J}, \mathcal{M})$ the schedule returned by $\mathcal{A}$ when run on $(\mathcal{J}, \mathcal{M})$. Similarly, $\mathcal{S}_{\mathrm{OPT}}(\mathcal{J}, \mathcal{M})$ denotes the optimal schedule, being $\mathrm{OPT}(\mathcal{J}, \mathcal{M})$ its minimum load. When it is clear from the context, we will drop the dependency on $\mathcal{J}$ or $\mathcal{M}$.

2.1 Algorithms with robust structure

An important fact used in the robust PTAS for makespan minimization from Sanders et al. [17] is that small jobs can be assigned greedily almost without affecting the approximation guarantee. This is however not the case for machine covering; see, e.g. [19] or [9]. A way to avoid this inconvenience is to develop algorithms that are oblivious to the arrival of small jobs, that is, algorithms where the assignment of big jobs is unaffected by arriving small job.

▶ **Definition 1.** Let $h \in \mathbb{R}_+$. An algorithm $\mathcal{A}$ has **robust structure at level h** if, for any instance $(\mathcal{J}, \mathcal{M})$ and $j^* \notin \mathcal{J}$ such that $p_{j^*} < h$, $\mathcal{S}_{\mathcal{A}}(\mathcal{J}, \mathcal{M})$ and $\mathcal{S}_{\mathcal{A}}(\mathcal{J} \cup \{j^*\}, \mathcal{M})$ assign to the same machines all the jobs in $\mathcal{J}$ with processing time at least h.

This definition highlights also the usefulness of working with the LPT rule, since the addition of a new small job to the instance does not affect the assignment of larger jobs. Indeed, it is easy to see the following.

▶ Remark. For any $h \in \mathbb{R}_+$, LPT has robust structure at level h.

We proceed now to define *relaxed* solutions where, roughly speaking, small jobs are added greedily on top of the assignment of big jobs.

▶ **Definition 2.** Let $\mathcal{A}$ be an α-approximation algorithm for the machine covering problem, with α constant, $k_1, k_2 \in \mathbb{R}_+$ constants, $1 \le k_1 \le k_2$ and $\varepsilon > 0$. Given a machine covering instance $(\mathcal{J}, \mathcal{M})$, a schedule $\mathcal{S}$ is a **(k_1, k_2)-relaxed version of $\mathcal{S}_{\mathcal{A}}$** if:

1. jobs with processing time at least $k_1\varepsilon\mathrm{OPT}$ are assigned exactly as in $\mathcal{S}_\mathcal{A}$, and
2. for every machine $i \in \mathcal{M}$, if $\mathcal{S}$ assigns at least one job of size less than $k_1\varepsilon\mathrm{OPT}$ to i, then $\ell_i(\mathcal{S}) \le \ell_{\min}(\mathcal{S}) + k_2\varepsilon\mathrm{OPT}$.

The following lemma shows that we can consider relaxed versions of known algorithms or solutions while almost not affecting the approximation factor. This will be helpful to control the migration of small jobs.

▶ **Lemma 3.** *Let $\mathcal{A}$ be an α-approximation, $\alpha \ge 1$ constant, $k_1, k_2 \in \mathbb{R}_+$ constants, $1 \le k_1 \le k_2$, $0 < \varepsilon < \frac{1}{2k_2\alpha}$ and $(\mathcal{J}, \mathcal{M})$ a machine covering instance. Every (k_1, k_2)-relaxed version of $\mathcal{S}_\mathcal{A}$ is an $(\alpha + O(\varepsilon))$-approximate solution.*

The described results allow us to significantly simplify the analysis of the designed algorithms. For example, consider LPT and suppose that at the arrival of jobs with processing time at least some specific value $h = \Theta(\varepsilon\mathrm{OPT})$ we can construct relaxed versions of solutions constructed by LPT. Dealing with an arriving job of size smaller than h becomes a simple task since assigning it to the current least loaded machine does not affect the assignment of big jobs, and we can prove that, for suitable constants k_1, k_2, a (k_1, k_2)-relaxed version of a solution constructed by LPT is maintained that way, almost preserving then its approximation ratio. It is important to remark that this approach is useful only if the algorithm has robust structure as, in general, the arrival of small jobs does not allow migration of big jobs and their structure may need to be changed because of these arrivals in order to maintain the approximation factor.

2.2 Rounding procedure

Another useful tool is rounding the processing times to simplify the instance and create symmetries while affecting the approximation factor only by a negligible value. Let us consider $0 < \varepsilon < 1$ such that $1/\varepsilon \in \mathbb{Z}$. We use the following rounding technique which is a slight modification of the one presented by Hochbaum and Shmoys in the context of makespan minimization on related machines [11]. For any job j, let $e_j \in \mathbb{Z}$ be such that $2^{e_j} \le p_j < 2^{e_j+1}$. We then round down p_j to the previous number of the form $2^{e_j} + k\varepsilon 2^{e_j}$ for $k \in \mathbb{N}$, that is, we define $\tilde{p}_j := 2^{e_j} + \left\lfloor \frac{p_j - 2^{e_j}}{\varepsilon 2^{e_j}} \right\rfloor \varepsilon 2^{e_j}$.

Observe that $p_j \ge \tilde{p}_j \ge p_j - \varepsilon 2^{e_j} \ge (1-\varepsilon)p_j$. Hence, an α-approximation algorithm for a rounded instance has an approximation ratio of $\alpha/(1-\varepsilon) = \alpha + O(\varepsilon)$ for the original instance. From now on we work exclusively with the rounded processing times.

Consider an upper bound UB on OPT such that $\mathrm{OPT} \le \mathrm{UB} \le 2\mathrm{OPT}$. This can be computed by any 2-approximation for the problem such as LPT. Consider the index set

$$\tilde{I}(\mathrm{UB}) := \left\{ i \in \mathbb{Z} : \varepsilon\mathrm{UB} \le 2^i < \mathrm{UB} \right\} = \{\ell, \dots, u\}. \tag{1}$$

We classify jobs as *small* if $\tilde{p}_j < 2^\ell$, *big* if $\tilde{p}_j \in [2^\ell, 2^{u+1})$, and *huge* otherwise. Notice that small jobs have size at most $2\varepsilon\mathrm{UB}$ and huge jobs have size at least UB. As we will see, our main difficulty will be given by big jobs; small and huge jobs are easy to handle. Notice that in every solution $\mathcal{S}$ constructed using LPT, if we ignore small jobs, huge jobs are assigned to a machine on their own and every machine $i \in \mathcal{M}$ without huge jobs has load at most 2UB. This is because i either has a big job alone, which has size at most 2UB, or it has load at most $\ell_{\min}(\mathcal{S}) + \tilde{p}_j \le 2\ell_{\min}(\mathcal{S}) \le 2\mathrm{UB}$, where j is the smallest job assigned to i. Let

$$\tilde{P} = \left\{ 2^i + k\varepsilon 2^i : i \in \{\ell, \dots, u\}, k \in \{0, 1, \dots, (1/\varepsilon) - 1\} \right\}, \tag{2}$$

be the set of all (rounded) processing times that a big job may take. The next lemma highlights the main properties of our rounding procedure.

▶ **Lemma 4.** *Consider the rounded job sizes $\tilde{p}_j$ for all j. Then it holds that,*

1. $|\tilde{P}| \in O((1/\varepsilon)\log(1/\varepsilon))$, *and*
2. *for each big and huge job j it holds that $\tilde{p}_j = h \cdot \varepsilon 2^{\ell}$ for some $h \in \mathbb{N}_0$.*

Unlike other standard rounding techniques (e.g. [19, 13]), the rounded sizes do not depend on OPT (or UB). This avoids possible migrations provoked by new rounded values, greatly simplifying our techniques.

3 A simple $(1.7+\varepsilon)$-competitive algorithm with $O(1/\varepsilon)$ migration.

In this section we will adapt a local search algorithm for Machine Covering to the online context with migration, using the properties of instances rounded as described in Section 2.2 to bound the migration factor.

In the context of online load balancing with migration, it is a good strategy to look for local search algorithms with good approximation guarantees and efficient running times. The main reason is that the migrated load corresponds to the sum of the migrated jobs in each local move, and for simplified instances (rounded, for example) the number of local moves until a locally optimal solution is found is usually a constant. That is the case for two natural neighborhoods used in local search algorithms for load balancing problems: *Jump* and *Swap*. Two solutions $\mathcal{S}, \mathcal{S}'$ are *jump-neighbors* if they assign the jobs to the same machines (up to relabeling of machines or jobs of equal size) except for at most one job, and *swap-neighbors* if they assign the jobs to the same machines (up to relabeling of machines or jobs of equal size) except for at most two jobs and, if they differ in exactly two jobs j_1, j_2 then they are in swapped machines, i.e., $\mathcal{S}(j_1) = \mathcal{S}'(j_2)$ and $\mathcal{S}(j_2) = \mathcal{S}'(j_1)$. The *weight* of a solution is defined through a two-dimensional vector having the minimum load of the schedule as first coordinate and the number of non-least loaded machines as second one. We compare the weight of two solutions lexicographically[1]. In other words, a solution is jump-optimal (respectively swap-optimal) if the migration of a single job (resp. the migration of a job or the swapping of two jobs) does not increase the minimum load and, if it maintains the minimum load, then it does not reduce the number of least loaded machines. The following lemma characterizes jump-optimal solutions for machine covering.

▶ **Lemma 5.** *Given $(\mathcal{J}, \mathcal{M})$ a machine covering instance, a schedule $\mathcal{S}$ is jump-optimal if and only if for any machine $i \in \mathcal{M}$ and any job $j \in \mathcal{S}^{-1}(i)$, we have that $\ell_i(\mathcal{S}) - p_j \leq \ell_{\min}(\mathcal{S})$.*

Chen et al. [3] proved tight bounds for the approximability of jump-optimal solutions. Their result is stated in a game theoretical framework, where jump-optimal solutions are equivalent to pure Nash equilibria for the Machine Covering game (see for example [20]). In this game, each job is a selfish agent trying to minimize the load of its own machine and the minimum load is the welfare function to be maximized. Through a small modification these bounds can be generalized to swap-optimal solutions as well (notice that a swap-optimal solution is jump-optimal by definition). We summarize the result in the following theorem which will be useful for our purposes.

[1] Just using the minimum load does not lead to good approximation ratios: think for example of $m > 2$ machines and m jobs of size 1; it is swap-optimal to assign all of them to the same machine.

Algorithm 1 Online jump-optimality.

Input: Instances $(\mathcal{J},\mathcal{M})$ and $(\mathcal{J}',\mathcal{M})$ such that $\mathcal{J}' = \mathcal{J} \cup \{j^*\}$; a schedule $\mathcal{S}(\mathcal{J},\mathcal{M})$.

```
1: run LPT on input $\mathcal{J}'$ and let $\tau$ be the minimum load. Set UB $\leftarrow 2\tau$. Define $\tilde{P}, \ell$, and $u$
   based on this upper bound UB using (1) and (2).
2: set $\mathcal{S}' \leftarrow \mathcal{S}$
3: if $\tilde{p}_{j^*} < 2^\ell$ then.                                ▷ Arriving job is small.
4:     assign $j^*$ to a least loaded machine in $\mathcal{S}'$.
5: else
6:     set $Q_B \leftarrow \{j^*\}$.                                  ▷ Set with unassigned big jobs.
7:     set $Q_s \leftarrow \emptyset$.                                ▷ Set with unassigned small jobs.
8:     while $Q_B \neq \emptyset$ do
9:         let $j$ be the largest job in $Q_B$. Set $Q_B \leftarrow Q_B \setminus \{j\}$.
10:        in $\mathcal{S}'_B$, use Push to assign $j$ to a least loaded machine $m^*$, obtaining its output
   set $Q$. Update $\mathcal{S}'_B$ to be the output solution of this procedure.
11:        reassign jobs in $\mathcal{S}'$ such that the assignment of (big) jobs in $\mathcal{S}'$ and $\mathcal{S}'_B$ coincides.
12:        while $m^*$ contains a small job w.r.t. UB and $\ell_{m^*}(\mathcal{S}') > \ell_{\min}(\mathcal{S}') + 2^\ell$ do
13:            remove the smallest job in $\mathcal{S}'^{-1}(m^*)$ and add it to $Q_s$.
14:        end while
15:        $Q_B \leftarrow Q_B \cup Q$.
16:    end while
17:    assign the jobs in $Q_s$ to $\mathcal{S}'$ using list-scheduling.
18: end if
19: return $\mathcal{S}'$.
```

▶ **Theorem 6** (from [3]). *Any locally optimal solution with respect to Jump (resp. Swap) for Machine Covering is* 1.7*-approximate. Moreover, there are instances showing that the approximation ratio of jump-(resp. swap-)optimality is at least* 1.7*.*

3.1 Online jump-optimality.

Using the rounding procedure from Section 2.2, jump-optimality can be adapted to the online context using a migration factor of $O\left(\frac{1}{\varepsilon}\right)$. Our algorithm, described in detail in Algorithm 1, is called every time a new job j^* arrives to the system, and receives as input the current solution $\mathcal{S}$ for $(\mathcal{J},\mathcal{M})$, initialized as empty if $\mathcal{J} = \emptyset$. It will output a (k,k)-relaxed version of a jump-optimal solution for some $k \leq 4$. We use the concept of a *list-scheduling* algorithm, that refers to assigning jobs iteratively (in any order) to some machine of minimum load. Given a schedule $\mathcal{S}$, $\mathcal{S}_B$ denotes the restriction of schedule $\mathcal{S}$ to big jobs.

The general idea of Algorithm 1 is to first round the instance, and assign the incoming job to a least loaded machine using an auxiliary algorithm called *Push* (see Algorithm 2). Push assigns a given job j into a given machine i and then iteratively removes the jobs in i that break jump-optimality according to Lemma 5, storing them in a set Q which is part of the output. Jobs removed by Push need to be reassigned, which we do by iteratively applying Push on each one of them which is big to assign them to the current least loaded machine until only small jobs are left to be assigned. At each iteration jump-optimality is preserved in a relaxed way, and as a last step all the unassigned small jobs are reassigned using list-scheduling. Notice that, since Push only removes jobs of size strictly smaller than the inserted job, each job is migrated at most once.

Algorithm 2 Push.

```
Input: Schedule $\mathcal{S}$ for $(\mathcal{J}, \mathcal{M})$, $i \in \mathcal{M}$, $j \notin \mathcal{J}$
Output: $Q \subseteq \mathcal{J}$, schedule $\mathcal{S}'$ for $((\mathcal{J} \cup \{j\}) \setminus Q, \mathcal{M})$
 1: $Q \leftarrow \emptyset$.
 2: $\mathcal{S}' \leftarrow \mathcal{S}$.
 3: assign $j$ to machine $i$ in $\mathcal{S}'$.
 4: for $k \in \mathcal{S}^{-1}(i)$ do
 5:     if $\ell_i(\mathcal{S}') - \tilde{p}_k > \ell_{\min}(\mathcal{S}')$ then
 6:         take out $k$ from $i$ in $\mathcal{S}'$.
 7:         $Q \leftarrow Q \cup \{k\}$.
 8:     end if
 9: end for
10: return $Q$, $\mathcal{S}'$.
```

▶ **Lemma 7.** *For any $h \in \mathbb{R}^+$, Algorithm 1 has robust structure at level h. Furthermore, Algorithm 1 is $(1.7 + O(\varepsilon))$-competitive and has polynomial running time.*

Proof idea. Robust structure of Algorithm 1 comes from the fact that Push removes jobs that are only smaller than the inserted job. We can then show that our solution is a $(k, 2k)$-relaxed version of a jump-optimal solution for $k = 2^\ell/(\varepsilon\mathrm{OPT}') \leq 4$, and we can conclude the first part of the result by using Theorem 6 and Lemma 3. Polynomial running time is implied by the fact that each job is migrated at most once. ◀

To analyze the migration factor, we define the *migration tree* of the algorithm as a node-weighted tree $G = (V, E)$, where V is the set of migrated jobs together with the incoming job $j^* \notin \mathcal{J}$, and the weight of each $v \in V$ is the processing time of the corresponding job $\tilde{p}_v$. The tree is constructed by first adding j^* as root. For each node (job) v in the tree, its children are defined as all the jobs migrated at the insertion of v. It is easy to see that this process does not create any loops as each job is migrated at most once. By definition, the leaves of the tree are the jobs not inducing migration, and thus any small job in the tree is a leaf. In the context of local search, the number of nodes in the tree corresponds to the number of iterations of the specific local search procedure. By analyzing the migration tree level by level, and together with the already discussed ideas, we can show the following result.

▶ **Lemma 8.** *Algorithm 1 uses migration factor $O((1/\varepsilon)\log(1/\varepsilon))$.*

Proof idea. Let w_i be the total processing time of jobs in level i of the migration tree. Every time a job j is inserted using Push, the total load of removed jobs in Q is strictly less than $\tilde{p}_j$, which means that w_i is strictly decreasing. Since w_i is strictly decreasing and jobs of size at most 2^ℓ do not induce migration, the tree has at most $|\tilde{P}| \in O((1/\varepsilon)\log(1/\varepsilon))$ levels, each of them having total load at most $\tilde{p}_{j^*}$. This implies that the total load of migrated big jobs is at most $O((1/\varepsilon)\log(1/\varepsilon)\tilde{p}_{j^*})$ and hence the migration factor is at most $O((1/\varepsilon)\log(1/\varepsilon))$. ◀

The analysis of the migration factor can be further refined to get a tight bound of $O(1/\varepsilon)$. The details can be found in the full version [9].

▶ **Theorem 9.** *Given $\varepsilon > 0$, Algorithm 1 is a polynomial time $(1.7+\varepsilon)$-competitive algorithm with migration factor $O(1/\varepsilon)$. Moreover, there are instances for which this factor is $\Omega(1/\varepsilon)$.*

4 LPT online with migration $\tilde{O}(1/\varepsilon^3)$.

In this section we present our main contribution which is an approximate online adaptation of LPT using poly$(1/\varepsilon)$ migration factor. In order to analyze it, we will first show some structural properties of the solutions constructed by LPT and how they behave when the instance is perturbed by a new job.

Algorithm 1 presented in Section 3 already gives some of the features and properties that our online version of LPT fulfills. However, now in the analysis we will crucially exploit the symmetry of instances rounded according to the procedure described in Section 2.2, in particular the fact that the load of each machine is a multiple of some fixed value. Since LPT takes decisions based solely on the machine loads, having a bounded number of values for them allows us to accurately control the set of machines where the assignment of big jobs can be kept unchanged after the arrival of a big job while maintaining the structure of the solution. Unless stated otherwise, for the rest of this section machine loads are considered with respect to the rounded processing times $\tilde{p}_j$.

Load Monotonicity. Here we describe in more detail the useful structural properties of solutions constructed using LPT.

▶ **Definition 10.** Given a schedule $\mathcal{S}$, its **load profile**, denoted by load$(\mathcal{S})$, is an $\mathbb{R}^m_{\geq 0}$-vector $(t_1, \ldots, t_m)$ containing the load of each machine sorted so that $t_1 \leq t_2 \leq \ldots \leq t_m$.

The following lemma shows that after the arrival of a job, the load profile of solutions constructed using LPT can only increase. This property only holds if the vector of loads is sorted, as it can be seen in Figure 1. This monotonicity property is essential for our analysis.

▶ **Lemma 11.** *Let $(\mathcal{J}, \mathcal{M})$ be a machine covering instance and $j^* \notin \mathcal{J}$ a job. Then, it holds that* load$(\mathcal{S}_{\text{LPT}}(\mathcal{J}, \mathcal{M})) \leq$ load$(\mathcal{S}_{\text{LPT}}(\mathcal{J}', \mathcal{M}))$, *where the inequality is considered coordinate-wise and $\mathcal{J}' = \mathcal{J} \cup \{j^*\}$.*

This lemma together with our rounding procedure allow us to show that the difference (in terms of the Hamming distance) of the load profiles of two consecutive solutions consisting purely of big jobs, is bounded by a small constant. This property will be important to obtain a poly$(1/\varepsilon)$ migration factor and here we crucially exploit the fact that the load of the machines is always multiple of a fixed value.

▶ **Lemma 12.** *Consider two instances $(\mathcal{J}, \mathcal{M})$ and $(\mathcal{J}', \mathcal{M})$ with $\mathcal{J}' = \mathcal{J} \cup \{j^*\}$, where $\mathcal{J}'$ contains only big or huge jobs w.r.t* UB. *Then the vectors* load$(\mathcal{S}_{\text{LPT}}(\mathcal{J}, \mathcal{M}))$ *and* load$(\mathcal{S}_{\text{LPT}}(\mathcal{J}', \mathcal{M}))$ *differ in at most $\frac{\tilde{p}_{j^*}}{\varepsilon 2^\ell} \in O(1/\varepsilon^2)$ many coordinates.*

Proof. Due to Lemma 11, we have that load$(\mathcal{S}_{\text{LPT}}(\mathcal{J}, \mathcal{M})) = (t_1, \ldots, t_m) \leq (t'_1, \ldots, t'_m) =$ load$(\mathcal{S}_{\text{LPT}}(\mathcal{J}', \mathcal{M}))$. Also, if $t_i < t'_i$ for some i, then $t'_i \geq t_i + \varepsilon 2^\ell$ since all values $t_j, t_{j'}$ are integer multiples of $\varepsilon 2^\ell$ because of Lemma 4. Since $||$load$(\mathcal{S}_{\text{LPT}}(\mathcal{J}', \mathcal{M})) -$ load$(\mathcal{S}_{\text{LPT}}(\mathcal{J}, \mathcal{M}))||_1 = \tilde{p}_{j^*}$, we obtain that the number of coordinates in which the load profiles differ is at most $\frac{\tilde{p}_{j^*}}{\varepsilon 2^\ell}$. Finally, recalling that j^* is big, then $\tilde{p}_{j^*} \leq 2^u \leq$ UB $\leq 2^\ell/\varepsilon$, and we can bound the number of different coordinates by $\frac{\tilde{p}_{j^*}}{\varepsilon 2^\ell} \leq 1/\varepsilon^2$. ◀

Description of Online LPT. Consider two instances $(\mathcal{J}, \mathcal{M})$ and $(\mathcal{J}', \mathcal{M})$ such that $\mathcal{J}' = \mathcal{J} \cup \{j^*\}$, and let OPT and OPT$'$ be their optimal values respectively. In what follows, for a given list-scheduling algorithm, we will refer to a tie-breaking rule as a rule that decides a particular machine for assigning a job when faced with multiple least loaded machines.

We say that an assignment is an LPT-solution if there is some tie-breaking rule such that LPT yields such assignment. We will compute an upper bound UB on OPT' by computing an LPT-solution and duplicating the value of its minimum load. For this upper bound, we compute its respective set $\tilde{P}$ with (1) and (2). In the algorithm, we will label elements in $\tilde{P} = \{q_1, \dots, q_{|\tilde{P}|}\}$ such that $q_1 > q_2 > \cdots > q_{|\tilde{P}|}$. Let $\mathcal{J}_h \subseteq \mathcal{J}$ (respectively $\mathcal{J}'_h \subseteq \mathcal{J}'$) be the set of jobs of size q_h in $\mathcal{J}$ (respectively $\mathcal{J}'$), for $q_h \in \tilde{P}$. Similarly, we define $\mathcal{J}_0$ (resp. $\mathcal{J}'_0$) to be the set of jobs in $\mathcal{J}$ (resp. $\mathcal{J}'$) of sizes larger than q_1, that is, all huge jobs in $\mathcal{J}$ (resp. $\mathcal{J}'$). Also, let $\mathcal{S}_h$ (resp. $\mathcal{S}'_h$) be the solution $\mathcal{S}$ (resp. $\mathcal{S}'$) restricted to jobs of size q_h or larger. Finally, $\mathcal{S}_0$ and $\mathcal{S}'_0$ are the respective solutions restricted to jobs in $\mathcal{J}_0$.

In what follows, x_+ denotes the positive part of $x \in \mathbb{R}$, i.e., $x_+ = \max\{x, 0\}$. To understand the algorithm, it is useful to have the following observation in mind.

▶ **Observation 13.** *Consider a solution $\mathcal{S}$ for jobs in $\mathcal{J}$ and let $\mathcal{K}$ be a set of jobs with $\mathcal{J} \cap \mathcal{K} = \emptyset$ and all jobs in $\mathcal{K}$ have the same size p. Consider a solution $\mathcal{S}_{LS}$ constructed by adding the jobs from $\mathcal{K}$ in $\mathcal{S}$ using list-scheduling, and let $\lambda = \ell_{\min}(\mathcal{S}_{LS})$. Notice that λ is independent of the tie-breaking rule used in list-scheduling. Consider any solution $\mathcal{S}'$ that is constructed starting from $\mathcal{S}$ and adding jobs in $\mathcal{K}$ in some arbitrary way. Then, $\mathcal{S}'$ corresponds to a solution obtained by adding jobs from $\mathcal{K}$ with a list-scheduling procedure (for some tie-breaking rule) if and only if the number of jobs in $\mathcal{K}$ added to each machine i is: (i) $\left\lceil \frac{(\lambda - \ell_i(\mathcal{S}))_+}{p} \right\rceil$ if $\frac{(\lambda - \ell_i(\mathcal{S}))_+}{p}$ is not an integer, and either $\frac{(\lambda - \ell_i(\mathcal{S}))_+}{p}$ or $\frac{(\lambda - \ell_i(\mathcal{S}))_+}{p} + 1$ if $\frac{(\lambda - \ell_i(\mathcal{S}))_+}{p}$ is a non-negative integer.*

Our main procedure is called every time that we get a new job j^* (where $\mathcal{J}' = \mathcal{J} \cup \{j^*\}$) and receives as input the current solution $\mathcal{S}$ for $(\mathcal{J}, \mathcal{M})$. If $\mathcal{J} = \emptyset$, then $\mathcal{S}$ is trivially initialized as empty. The exact description is given in Algorithm 3.

Broadly speaking, the algorithm works in phases $h \in \{0, \dots, |\tilde{P}|\}$, where for each h it assigns jobs in $\mathcal{J}'_h$. First, we assign jobs exactly as in $\mathcal{S}_h$ for machines in which the assignment of $\mathcal{S}_{h-1}$ and $\mathcal{S}'_{h-1}$ coincide. The set of such machines is denoted by $\mathcal{M}^{=}_{h-1}$ and the set of remaining machines is denoted by $\mathcal{M}^{\neq}_{h-1}$. As we will see, this is consistent with LPT by the previous observation and Lemma 11. The remaining jobs in $\mathcal{J}'_h$ are assigned using list-scheduling. Crucially, we will break ties in favor of machines where the assignment of $\mathcal{S}_{h-1}$ and $\mathcal{S}'_{h-1}$ differ. This is necessary to avoid creating new machines with different assignments. After assigning huge and big jobs, small jobs are added exactly as in $\mathcal{S}$ in machines where the assignment of big jobs in $\mathcal{S}$ and $\mathcal{S}'$ coincides. The rest of small jobs are added greedily. In the last part, the algorithm rebalances small jobs by moving them from machines of load higher than $\ell_i(\mathcal{S}') + 2^\ell$ to the least loaded machines.

We can prove the following lemma in a very similar way to Lemma 7.

▶ **Lemma 14.** *Algorithm 3 is $(4/3 + O(\varepsilon))$-competitive.*

Bounding the migration factor. To analyze the migration factor of the algorithm, we will show that $|\mathcal{M}^{\neq}_{|\tilde{P}|}|$ is upper bounded by a constant. This will be done inductively by first bounding $|\mathcal{M}^{\neq}_h \setminus \mathcal{M}^{\neq}_{h-1}|$ for each h and then using the fact that $|\tilde{P}| \in O((1/\varepsilon)\log(1/\varepsilon))$. A description of the overall idea can be found in Figure 2.

Let us consider huge jobs w.r.t UB (i.e. jobs in $\mathcal{J}'_0$). Notice that all these jobs are larger than $\mathrm{OPT}' \geq \mathrm{OPT}$, and hence in $\mathcal{S}'_0$ each one is assigned alone to one machine. The same situation happens in solution $\mathcal{S}$ restricted to jobs in $\mathcal{J}_0$. Thus, none of these jobs are migrated. Hence, we can assume w.l.o.g. for the sake of the analysis of the migration that all jobs are big or small w.r.t UB (including j^*). Additionally, we can assume that j^* is not small, since otherwise there is no migration.

Algorithm 3 Online LPT.

```
Input: Instances $(\mathcal{J},\mathcal{M})$ and $(\mathcal{J}',\mathcal{M})$ such that $\mathcal{J}' = \mathcal{J} \cup \{j^*\}$; a schedule $\mathcal{S}(\mathcal{J},\mathcal{M})$.
 1: run LPT on input $\mathcal{J}'$ and let $\tau$ be the minimum load of the constructed solution. Set
    UB $\leftarrow 2\tau$. Define $\tilde{P}, \ell$, and $u$ based on this upper bound UB using (1) and (2).
 2: set $\mathcal{M}^{=}_{-1} \leftarrow \mathcal{M}$ and $\mathcal{M}^{\neq}_{-1} \leftarrow \emptyset$.
 3: for $h = 0, 1, \ldots, |\tilde{P}|$ do                                  ▷ Assignment of big and huge jobs
 4:     for each machine $i \in \mathcal{M}^{=}_{h-1}$, assign all jobs in $\mathcal{J}_h \cap \mathcal{S}^{-1}(i)$ to $i$ in $\mathcal{S}'$.
 5:     for jobs in $\mathcal{J}'_h$ still not assigned in $\mathcal{S}'$, apply list-scheduling (with an arbitrary order
        of jobs). If there is more than one least loaded machine break ties in favor of $\mathcal{M}^{\neq}_{h-1}$.
 6:     define $\mathcal{M}^{=}_h$ as the set of machines $i$ such that $\mathcal{S}^{-1}_h(i) = \mathcal{S}'^{-1}_h(i)$ and $\mathcal{M}^{\neq}_h \leftarrow \mathcal{M} \setminus \mathcal{M}^{=}_h$.
 7: end for
 8: for machines $i \in \mathcal{M}^{=}_{|\tilde{P}|}$ do                                     ▷ Assignment of small jobs
 9:     assign all small jobs w.r.t to UB in $\mathcal{J} \cap \mathcal{S}^{-1}(i)$ to $i$ in $\mathcal{S}'$.
10: end for
11: assign the remaining jobs using list-scheduling.
12: set $\overline{\mathcal{M}}$ to be the set of machines containing a small job w.r.t UB.
13: while there exists $i \in \overline{\mathcal{M}}$ s.t. $\ell_i(\mathcal{S}') > \ell_{\min}(\mathcal{S}') + 2^{\ell}$ do
14:     consider a machine $i \in \overline{\mathcal{M}}$ of maximum load. Reassign the smallest job in $\mathcal{S}'^{-1}(i)$ to
        any least loaded machine.
15:     update $\overline{\mathcal{M}}$ to be the set of machines containing a small job w.r.t UB.
16: end while
17: return $\mathcal{S}'$.
```

Let $\mathcal{J}^{=}_h$ be the set of jobs assigned by Step 5 to machines in $\mathcal{M}^{=}_{h-1}$. Notice that the jobs in $\mathcal{J}^{=}_h$ correspond to the jobs in $\mathcal{J}'_h$ that $\mathcal{S}'$ assigns to a machine in $\mathcal{M}^{=}_{h-1}$ but $\mathcal{S}$ processes in $\mathcal{M}^{\neq}_{h-1}$. The next lemma is the main technical contribution of this section.

▶ **Lemma 15.** *For all $h \in \{1, \ldots, |\tilde{P}|\}$ it holds that $|\mathcal{M}^{\neq}_h \setminus \mathcal{M}^{\neq}_{h-1}| \in O(\frac{\tilde{p}_{j^*}}{\varepsilon 2^{\ell}})$.*

The strategy to prove this lemma is first to show that $|\mathcal{J}^{=}_h| \in O(\frac{\tilde{p}_{j^*}}{\varepsilon 2^{\ell}})$; this is the main difficulty and for the proof we use lemmas 11 and 12. Having this, since jobs in $\mathcal{J}^{=}_h$ are the only jobs assigned in a given iteration h that can cause one new machine to have different assignments in $\mathcal{S}_h$ and $\mathcal{S}'_h$, then $|\mathcal{M}^{\neq}_h \setminus \mathcal{M}^{\neq}_{h-1}| \leq |\mathcal{J}^{=}_h|$ and the lemma holds.

Let $\mathcal{S}_{\mathrm{LPT},h}$ be an LPT-solution for jobs in $\mathcal{J}_0 \cup \ldots \cup \mathcal{J}_h$, and similarly $\mathcal{S}'_{\mathrm{LPT},h}$ for jobs in $\mathcal{J}'_0 \cup \ldots \cup \mathcal{J}'_h$. Let us fix $h \geq 1$ and consider the target values $\lambda = \ell_{\min}(\mathcal{S}_{\mathrm{LPT},h})$ and $\lambda' = \ell_{\min}(\mathcal{S}'_{\mathrm{LPT},h})$. Notice that by Lemma 11, $\lambda \leq \lambda'$. In order to bound $|\mathcal{J}^{=}_h|$, we first show in the following lemma that, if a job is actually assigned by Step 5 to some machine in $\mathcal{M}^{=}_{h-1}$, then many jobs from the stage must be assigned to machines in $\mathcal{M}^{\neq}_{h-1}$.

▶ **Lemma 16.** *Assume that $\mathcal{J}^{=}_h \neq \emptyset$. For each machine $i \in \mathcal{M}^{\neq}_{h-1}$, if $\lambda - \ell_i(\mathcal{S}'_{h-1}) \geq 0$ solution $\mathcal{S}'_h$ assigns to i at least $\left\lfloor \frac{(\lambda - \ell_i(\mathcal{S}'_{h-1}))_+}{q_h} \right\rfloor + 1$ many jobs from $\mathcal{J}_h$.*

Now we can sketch the proof that $|\mathcal{J}^{=}_h| \in O(\frac{\tilde{p}_{j^*}}{\varepsilon 2^{\ell}})$ (a detailed proof can be found in [9]).

▶ **Lemma 17.** *It holds that $|\mathcal{J}^{=}_h| \in O(\frac{\tilde{p}_{j^*}}{\varepsilon 2^{\ell}})$.*

Proof Sketch. Assume w.l.o.g. that $\mathcal{M}^{\neq}_{h-1} = \{1, \ldots, m'\}$ and that $\ell_1(\mathcal{S}'_{h-1}) \leq \ell_2(\mathcal{S}'_{h-1}) \leq \cdots \leq \ell_{m'}(\mathcal{S}'_{h-1})$. Consider also a permutation $\sigma : \mathcal{M}^{\neq}_{h-1} \to \mathcal{M}^{\neq}_{h-1}$ such that $\ell_{\sigma(1)}(\mathcal{S}_{h-1}) \leq$

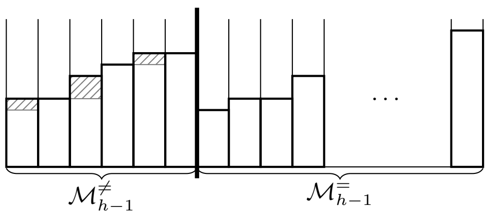


Figure 2 Depiction of a possible situation at the end of iteration $h-1$. The machines on the right side correspond to machines in $\mathcal{M}^{=}_{h-1}$ and therefore process the same jobs in $\mathcal{S}_{h-1}$ and $\mathcal{S}'_{h-1}$. Assume, possibly erroneously and just as a thought experiment, that the machines in $\mathcal{M}^{\neq}_{h-1}$ can be sorted non-decreasingly by load for $\mathcal{S}_{h-1}$ and $\mathcal{S}'_{h-1}$ simultaneously. The two solutions are depicted simultaneously in the picture, where the difference of loads on machines in $\mathcal{M}^{\neq}_{h-1}$ corresponds to the dashed area. The total dashed load equals to $\tilde{p}_{j^*}$, which is spread in only constantly many machines by Lemma 12. When assigning jobs in $\mathcal{J}_h$, the algorithm first assigns a number of jobs to each machine in $\mathcal{M}^{=}_{h-1}$ (Step 4), and then fills machines in $\mathcal{M}^{\neq}_{h-1}$. Notice that while the algorithm does not assign another job to a machine in $\mathcal{M}^{=}_{h-1}$, no new machine will enter $\mathcal{M}^{\neq}_{h} \setminus \mathcal{M}^{\neq}_{h-1}$. On the other hand, the number of such jobs can be bounded by a number proportional to $\tilde{p}_{j^*}$ (and $1/\varepsilon$), which then also bounds the number of machines in $\mathcal{M}^{\neq}_{h} \setminus \mathcal{M}^{\neq}_{h-1}$. In reality, however, it is not true that the machines in $\mathcal{M}^{\neq}_{h-1}$ can be sorted non-decreasingly on the loads for $\mathcal{S}_{h-1}$ and $\mathcal{S}'_{h-1}$ simultaneously. This provokes a number of technical difficulties that we avoid by using a different permutation of machines for each solution and invoking Lemma 11.

$\ell_{\sigma(2)}(\mathcal{S}_{h-1}) \leq \cdots \leq \ell_{\sigma(m')}(\mathcal{S}_{h-1})$. By using Lemma 11, we can show that $\ell_{\sigma(i)}(\mathcal{S}_{h-1}) \leq \ell_i(\mathcal{S}'_{h-1})$ for all $i \in \mathcal{M}^{\neq}_{h-1}$. Let us consider sets

$$\begin{aligned} T_- &= \{i \in \mathcal{M}^{\neq}_{h-1} : \ell_i(\mathcal{S}'_{h-1}) \leq \lambda\}, \text{ and} \\ T_+ &= \{i \in \mathcal{M}^{\neq}_{h-1} : \ell_{\sigma(i)}(\mathcal{S}_{h-1}) \leq \lambda \text{ and } \ell_i(\mathcal{S}'_{h-1}) > \lambda\}. \end{aligned}$$

Lemma 16 implies that the total number of jobs from $\mathcal{J}'_h$ assigned by $\mathcal{S}'_h$ to machines in $\mathcal{M}^{\neq}_{h-1}$ is at least

$$\sum_{i \in T_-} \left(\left\lfloor \tfrac{(\lambda - \ell_i(\mathcal{S}'_{h-1}))_+}{q_h} \right\rfloor + 1 \right) = \sum_{i \in T_- \cup T_+} \left(\left\lfloor \tfrac{(\lambda - \ell_{\sigma(i)}(\mathcal{S}_{h-1}))_+}{q_h} \right\rfloor + 1 \right)$$
$$- \sum_{i \in T_+} \left(\left\lfloor \tfrac{(\lambda - \ell_{\sigma(i)}(\mathcal{S}_{h-1}))_+}{q_h} \right\rfloor + 1 \right) + \sum_{i \in T_-} \left\lfloor \tfrac{(\lambda - \ell_i(\mathcal{S}'_{h-1}))_+}{q_h} \right\rfloor - \left\lfloor \tfrac{(\lambda - \ell_{\sigma(i)}(\mathcal{S}_{h-1}))_+}{q_h} \right\rfloor .$$

Since $T_- \cup T_+$ contains all indices $i \in \mathcal{M}^{\neq}_{h-1}$ such that $\ell_{\sigma(i)}(\mathcal{S}_{h-1}) \leq \lambda$, we have that $\sum_{i \in T_- \cup T_+} \left(\left\lfloor \tfrac{(\lambda - \ell_{\sigma(i)}(\mathcal{S}_{h-1}))_+}{q_h} \right\rfloor + 1 \right) \geq |\mathcal{J}^{=}_h| - 1$. With a bit of work we get that

$$|\mathcal{J}^{=}_h| \leq 1 + |T_+| + \sum_{i \in T_+} \left\lfloor \tfrac{(\lambda - \ell_i(\mathcal{S}'_{h-1}))_+}{q_h} \right\rfloor + \sum_{i \in T_- \cup T_+} \left\lfloor \tfrac{(\lambda - \ell_{\sigma(i)}(\mathcal{S}_{h-1}))_+}{q_h} \right\rfloor - \left\lfloor \tfrac{(\lambda - \ell_i(\mathcal{S}'_{h-1}))_+}{q_h} \right\rfloor ,$$

which can be simplified even more since $\sum_{i \in T_+} \left\lfloor \tfrac{(\lambda - \ell_i(\mathcal{S}'_{h-1}))_+}{q_h} \right\rfloor = 0$. Finally, if we consider $T_{\neq} = \{i \in \mathcal{M}^{\neq}_{h-1} : \ell_{\sigma(i)}(\mathcal{S}_{h-1}) \neq \ell_i(\mathcal{S}'_{h-1})\}$, the last expression is at most

$$\begin{aligned} |\mathcal{J}^{=}_h| &\leq 1 + |T_+| + \sum_{i \in (T_- \cup T_+) \cap T_{\neq}} \left\lfloor \tfrac{(\lambda - \ell_{\sigma(i)}(\mathcal{S}_{h-1}))_+}{q_h} \right\rfloor - \left\lfloor \tfrac{(\lambda - \ell_i(\mathcal{S}'_{h-1}))_+}{q_h} \right\rfloor \\ &\leq 1 + |T_+| + |T_{\neq}| + \sum_{i \in T_{\neq}} \tfrac{\ell_i(\mathcal{S}'_{h-1}) - \ell_{\sigma(i)}(\mathcal{S}_{h-1})}{q_h}, \end{aligned}$$

which concludes the proof since $|T_{\neq}| \leq \frac{\tilde{p}_{j^*}}{\varepsilon 2^\ell}$ (Lemma 12) and the last sum is at most $\frac{\tilde{p}_{j^*}}{\varepsilon 2^\ell}$. ◀

▶ **Theorem 18.** *Online LPT is a polynomial time $(4/3 + O(\varepsilon))$-competitive algorithm with $O((1/\varepsilon^3)\log(1/\varepsilon))$ migration factor.*

We complement this result by improving the lower bound on the best possible competitive ratio for an algorithm with constant migration factor (details can be found in [9]).

▶ **Lemma 19.** *For any $\varepsilon > 0$, there is no $\left(\frac{17}{16} - \varepsilon\right)$-competitive algorithm using constant migration factor for the online machine covering problem with migration.*

References

1 Y. Azar and L. Epstein. On-line machine covering. *J. Sched.*, 1:67–77, 1998.

2 S. Berndt, K. Jansen, and K. Klein. Fully dynamic bin packing revisited. In *APPROX/RANDOM 2015*, pages 135–151, 2015.

3 Xujin Chen, Leah Epstein, Elena Kleiman, and Rob van Stee. Maximizing the minimum load: The cost of selfishness. *Theor. Comput. Sci.*, 482:9–19, 2013.

4 J. Csirik, H. Kellerer, and G. Woeginger. The exact LPT-bound for maximizing the minimum completion time. *Oper. Res. Lett.*, 11:281–287, 1992.

5 B. Deuermeyer, D. Friesen, and M. Langston. Scheduling to maximize the minimum processor finish time in a multiprocessor system. *SIJADM*, 3:190–196, 1982.

6 L. Epstein and A. Levin. A robust APTAS for the classical bin packing problem. *Math. Program.*, 119:33–49, 2009.

7 L. Epstein and A. Levin. Robust algorithms for preemptive scheduling. *Algorithmica*, 69:26–57, 2014.

8 A. Frangioni, E. Necciari, and M. Scutellà. A multi-exchange neighborhood for minimum makespan parallel machine scheduling problems. *J. Comb. Optim.*, 8:195–220, 2004.

9 Waldo Gálvez, José A. Soto, and José Verschae. Symmetry exploitation for online machine covering with bounded migration. *CoRR*, 2016. `arXiv:1612.01829`.

10 A. Gu, A. Gupta, and A. Kumar. The power of deferral: Maintaining a constant-competitive steiner tree online. *SIAM J. Comput.*, 45:1–28, 2016.

11 D. Hochbaum and D. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17:539–551, 1988.

12 K. Jansen and K. Klein. A robust AFPTAS for online bin packing with polynomial migration. In *ICALP 2013*, pages 589–600, 2013.

13 K. Jansen, K. Klein, and J. Verschae. Closing the gap for makespan scheduling via sparsification techniques. In *ICALP 2016*, pages 1–13, 2016.

14 J. Łacki, J. Oćwieja, M. Pilipczuk, P. Sankowski, and A. Zych. The power of dynamic distance oracles: Efficient dynamic algorithms for the steiner tree. In *STOC 2015*, pages 11–20, 2015.

15 N. Megow, M. Skutella, J. Verschae, and A. Wiese. The power of recourse for online MST and TSP. *SIAM J. Comput.*, 45:859–880, 2016.

16 D. Recalde, C. Rutten, P. Schuurman, and T. Vredeveld. Local Search Performance Guarantees for Restricted Related Parallel Machine Scheduling. *LATIN 2010*, pages 108–119, 2010.

17 P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. *Math. Oper. Res.*, 34:481–498, 2009.

18 P. Schuurman and T. Vredeveld. Performance guarantees of local search for multiprocessor scheduling. *INFORMS J. Comput.*, 19:52–63, 2007.

19 M. Skutella and J. Verschae. Robust polynomial-time approximation schemes for parallel machine scheduling with job arrivals and departures. *Math. Oper. Res.*, 41:991–1021, 2016.

20 B. Vöcking. Selfish load balancing. In *Algorithmic Game Theory*, pages 517–542. Cambridge University Press, 2007.

21 G. Woeginger. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Oper. Res. Lett.*, 20:149–154, 1997.

Edit Distance with Block Operations

Michał Gańczorz
Institute of Computer Science, University of Wrocław, Poland
mga@cs.uni.wroc.pl

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland
gawry1@gmail.com

Artur Jeż
Institute of Computer Science, University of Wrocław, Poland
aje@cs.uni.wroc.pl

Tomasz Kociumaka
Institute of Informatics, University of Warsaw, Poland
kociumaka@mimuw.edu.pl

Abstract

We consider the problem of edit distance in which block operations are allowed, i.e. we ask for the minimal number of (block) operations that are needed to transform a string s to t. We give $\mathcal{O}(\log n)$ approximation algorithms, where n is the total length of the input strings, for the variants of the problem which allow the following sets of operations: block move; block move and block delete; block move and block copy; block move, block copy, and block uncopy. The results still hold if we additionally allow any of the following operations: character insert, character delete, block reversal, or block involution (involution is a generalisation of the reversal). Previously, algorithms only for the first and last variant were known, and they had approximation ratios $\mathcal{O}(\log n \log^* n)$ and $\mathcal{O}(\log n (\log^* n)^2)$, respectively. The edit distance with block moves is equivalent, up to a constant factor, to the common string partition problem, in which we are given two strings s, t and the goal is to partition s into minimal number of parts such that they can be permuted in order to obtain t. Thus we also obtain an $\mathcal{O}(\log n)$ approximation for this problem (compared to the previous $\mathcal{O}(\log n \log^* n)$).

The results use a simplification of the previously used technique of locally consistent parsing, which groups short substrings of a string into phrases so that similar substrings are guaranteed to be grouped in a similar way. Instead of a sophisticated parsing technique relying on a deterministic coin tossing, we use a simple one based on a partition of the alphabet into two subalphabets. In particular, this lowers the running time from $\mathcal{O}(n \log^* n)$ to $\mathcal{O}(n)$. The new algorithms (for block copy or block delete) use a similar algorithm, but the analysis is based on a specially tuned combinatorial function on sets of numbers.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Edit distance, Block operations, Common string partition

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.33

Funding The work of the first and third author was supported under National Science Centre, Poland, project number 2014/15/B/ST6/00615. The work of the fourth author was supported under National Science Centre, Poland, project number number 2014/13/B/ST6/00770.

© Michał Gańczorz, Paweł Gawrychowski, Artur Jeż, and Tomasz Kociumaka;
licensed under Creative Commons License CC-BY
26th Annual European Symposium on Algorithms (ESA 2018).
Editors: Yossi Azar, Hannah Bast, and Grzegorz Herman; Article No. 33; pp. 33:1–33:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the *edit distance problem*, which is one of the most iconic problems in the field of string algorithms, we are given two strings and a set of allowed operations, and we ask for the minimum number of operations needed to transform one of the strings into the other. Classically, we allow single-letter operations (usually: character insert, delete, and replace), but it seems that block operations, in which the whole substrings of the input can be edited in one operation, are as important and practical.

In the classical setting, when character operations are allowed, edit distance is computable in quadratic time, and achieving strongly subquadratic time is unlikely [2]. Allowing block deletion does not make the problem substantially harder, and a polynomial-time algorithm for this variant is known [21, 20].

The variant with other block operations was first considered by Lopresti and Tomkins [13], who showed NP-hardness of edit distance with block moves, as well as with block moves and block deletions. The former problem was approximated within an $\mathcal{O}(\log n \operatorname{poly}(\log^* n))$ factor [7], which was later improved to $\mathcal{O}(\log n \log^* n)$ [6]. A slightly worse approximation ratio of $\mathcal{O}(\log n(\log^* n)^2)$ is known when we allow block move, block copying, block uncopying and block reversal [15, 16]; while all those algorithms did not explicitly allow character edits (insert, delete, replace), it is clear from their analysis that those can also be accommodated. A variant with block move and block delete was considered and some structural properties were shown [20], but in the end no approximation algorithm was given.

The three mentioned approximation algorithms are all based on the locally sensitive parsing technique, which has roots in the deterministic coin tossing by Cole and Vishkin [5] and was used previously in the context of string algorithms in general [18] and comparing strings in particular [17, 14, 1]. In this method, we partition the string into constant-length blocks such that for each letter we can decide whether it begins or ends a block based only on the $\mathcal{O}(\log^* n)$-size neighbourhood of this letter. Then we label the blocks with new symbols and iterate the process. It turns out that to approximate the edit distance between two strings, it is enough to count the difference between the numbers of labels that appear during this (iterated) process; this is turn can be abstracted as calculating the ℓ_1 norm between embeddings into a vector space.

Surprisingly, allowing *both* block deletion and block copy makes approximation of the edit distance simpler: there are $\mathcal{O}(1)$ approximation algorithms for this problem [8, 19]. Those are based on a different approach, though: in essence they parse the target into phrases using the LZ77 algorithm, copy the phrases from the source, and then delete the source.

The edit distance with move operations problem is equivalent (up to a constant coefficient) to a *common string partition problem*, which was investigated on its own due to its connections with the computational biology, however, often in variants that are not so well motivated in terms of edit distance. For instance, it was shown to be fixed parameter tractable [3] and its restricted variant is known to be NP-hard but at the same time approximable up to a constant factor [10]; heuristics for this problem were also analysed [4].

Our contribution. We present $\mathcal{O}(\log n)$ approximation algorithms for the edit distance problem with the following set of (block) operations: block move; block move and block delete; block move and block copy; block move, block copy and block uncopy. Our algorithms work also when an arbitrary subset of the following operations is also allowed: character insert, character delete, character replace, block involution. (Involution, also known as antimorphism, is a generalisation of reverse: it reverses the string and then replaces each letter a with

$f(a)$, where f is a given bijection on the letters such that $f(f(a)) = a$, note that f can be the identity.) The first algorithm improves upon the previously known $\mathcal{O}(\log n \log^* n)$ approximation ratio [6], while the last one – the $\mathcal{O}(\log n (\log^*)^2 n)$ ratio [15, 16]. The second variant was considered to no avail [20]; to the best of our knowledge, the third variant has not been considered before.

The algorithms for the cases when only block move or block move, copy and uncopy are allowed, are similar as before [6, 15, 16], but instead of the sophisticated locally consistent parsing based on the deterministic coin tossing, we use a simpler one which is based on a partition of the alphabet into two parts. Such approaches were recently investigated [11, 9].

The presented version of the parsing is much simpler than previously used and allows for the removal of the multiplicative $\log^* n$ factors from the approximation ratios. It also enables a more general treatment of involution instead of reversal.

The algorithm for block moves and block delete is almost the same as in the case when only block moves are allowed. However, the analysis employs complex combinatorial functions defined on sets of lengths of letter repetitions. Unlike the previously used embedding to ℓ_1 spaces, this function depends on *both* strings and cannot be computed separately for each of them. The algorithm for the variant with block move and block copy uses the same function in the analysis, but in contrast to other presented algorithms (as well as the previously known ones), it is no longer a simple greedy algorithm. It constructs the sequence of operations in two steps: in the second one, the earlier copy operations may be revoked and move operations may be forced.

Our algorithms can be generalised to the case when the input is given in a grammar-compressed form: then its running time becomes $\mathcal{O}(n \log N)$, when n is the compressed size of the input and N the sum of lengths of the decompressed strings.

To streamline the presentation, in the extended abstract we give the algorithms in the variant when the involution is not allowed. The generalisation to the case with involution is natural, though tedious.

2 Definitions and basic reductions

A *string* is a sequence of elements, called *letters*, from a finite set, called *alphabet* and usually denoted by Σ, and it is denoted as $w = w_1 w_2 \cdots w_k$, where each w_i is a letter; the *length* $|w|$ of such a string w is k. For any two strings $w = w_1 \cdots w_k$ and $w' = w_{k+1} \cdots w_{k+\ell}$, their *concatenation* is $ww' = w_1 \cdots w_{k+\ell}$. A string v is a *substring* of w if there exist strings w', w'' such that $w = w'vw''$, it is a *prefix* if $w = vw''$ and a *suffix* if $w = w'v$. The empty string, i.e. the one of length 0, is denoted by ϵ. For a letter a and a string w, the number of occurrences of a in w is denoted $|w|_a$.

Given two strings s, t their *edit distance* is the minimum number of operations needed to transform s to t. The usual operations are insert (ins) and delete (del): the former turns a string $s = s_1 s_2$ to $s_1 a s_2$ and the latter $s' = s_1 a s_2$ to $s_1 s_2$ for arbitrary letter a and strings s_1, s_2. Replace, which replaces a single letter with another, is usually considered as well, but it can be simulated by insert and delete, so we ignore it later on. Other operations include block copy (called copy for short, cp), block move (called move for short, mv) and block delete ($\mathsf{b\text{-}del}$), which can transform $s = s_1 s_2 s_3 s_4$ to, respectively, $s_1 s_2 s_3 s_2 s_4$ or $s_1 s_3 s_2 s_3 s_4$, $s_1 s_3 s_2 s_4$ and $s_1 s_3 s_4$, for arbitrary strings s_1, s_2, s_3, s_4. The block uncopy (called uncopy for short, uncp) is the inverse operation to copy, i.e. it can transform any $s_1 s_2 s_3 s_2 s_4$ to $s_1 s_2 s_3 s_4$ or $s_1 s_3 s_2 s_4$ for arbitrary strings $s_1, \ldots, s_4$. By $\mathrm{ED}_{\mathsf{Op}}(s, t)$ we denote the minimal number of operations from the set Op that transform s to t, where $\mathsf{Op} \subseteq \{\mathsf{ins}, \mathsf{del}, \mathsf{cp}, \mathsf{mv}, \mathsf{uncp}, \mathsf{b\text{-}del}\}$, and the edit distance with operations Op problem asks, for given strings s and t, to find the sequence of $\mathrm{ED}_{\mathsf{Op}}(s,t)$ operations that transforms s to t. Note that the "edit distance" is a distance only when block deletion is not allowed and for each operation its inverse is also allowed. Nevertheless, in each case ED does satisfy the *(directed) triangle inequality*: $\mathrm{ED}_{\mathsf{Op}}(s,t) + \mathrm{ED}_{\mathsf{Op}}(t,\ell) \geq \mathrm{ED}_{\mathsf{Op}}(s,\ell)$. Still, we use the name *distance* for historic reasons.

For two strings s and t, their *common partition with operations* is a representation $s = s_1 s_2 \cdots s_{d_s}$ and $t = t_1 t_2 \cdots t_{d_t}$ with two sets of indices $I_s \subseteq [1\,..\,d_s]$ and $I_t \subseteq [1\,..\,d_t]$ (equal to $[1\,..\,d_s]$ and $[1\,..\,d_t]$, respectively, unless otherwise stated), and a bijection $f : I_s \to I_t$ such that $s_i = t_{f(i)}$ for each $i \in I_s$; we say that parts s_i and $t_{f(i)}$ are *matched.* The *size* of such a partition is $d_s + d_t$. Depending on the allowed operations, we may relax some of those requirements and give new ones:

delete If deletion of single letters is allowed (del), then we allow $I_s \neq [1\,..\,d_s]$ but require that $|s_i| = 1$ for $i \notin I_s$. We say that such letters are *deleted.*

insert If insertion of single letters is allowed (ins), then we allow $I_t \neq [1\,..\,d_t]$ but require that $|t_i| = 1$ for $i \notin I_t$. We say that such letters are *inserted.*

block-delete If block-deletion is allowed ($\mathsf{b\text{-}del}$), then we allow $I_s \neq [1\,..\,d_s]$. This operation supersedes deletion. We say that such blocks are *deleted.*

By $\mathrm{CP}_{\mathsf{Op}}(s,t)$ for $\mathsf{Op} \subseteq \{\mathsf{del}, \mathsf{ins}, \mathsf{b\text{-}del}\}$, we denote the minimal size of the common partition with operations Op for s and t. In the minimum common string partition with operations Op problem, we want to compute, for the given strings s and t, their partition of minimal size and the corresponding function f.

Note that the different names for deletion and insertion of letters are chosen for consistency between the common partition and the edit distance problems. In the later sections, we will consider a common string partition (without operations) problem generalised to two sets of strings, which is defined in the obvious way.

It is folklore knowledge that edit distance with move operations corresponds to a common partition; more precisely, it is within constant factor of the minimal common partition. Moreover, the same holds when block deletion and/or character operations are allowed.

▶ **Lemma 1.** *For any set of operations* $\mathsf{Op} \subseteq \{\mathsf{del}, \mathsf{ins}, \mathsf{b\text{-}del}\}$*, there is a constant* c_{Op} *such that for any strings* s, t*:*

$$\mathrm{ED}_{\{\mathsf{mv}\}\cup\mathsf{Op}}(s,t) \leq \mathrm{CP}_{\mathsf{Op}}(s,t) \leq c_{\mathsf{Op}}\,\mathrm{ED}_{\{\mathsf{mv}\}\cup\mathsf{Op}}(s,t) \quad .$$

Moreover, this correspondence is effective: given a sequence of d operations from Op *that transform s to t, we can compute the common partition with* Op *of s and t of size at most $c_{\mathsf{Op}}d$, and given a common partition with* Op *of size d, we can compute a sequence of d operations from* Op *that transform s to t.*

As approximation algorithms given in this work have approximation factors $\mathcal{O}(\log|st|)$, due to Lemma 1 we will content ourselves with considering one or the other problem of edit distance or common partition, depending on whichever is easier to argue about.

3 Locally consistent parsing

A *parsing* of a string s is a sequence $s_1, \ldots, s_k$ such that $s = s_1 s_2 \cdots s_k$; the strings $s_1, \ldots, s_k$ are called *phrases*, the integer k is the *size* of this parsing, and we say that s is *parsed* into $s_1, \ldots, s_k$. Given a substring t of s, we say that it is *parsed into* $s_i, \ldots, s_j$ when $s_i \cdots s_j$ contain this occurrence of t while $s_{i+1} \cdots s_j$ and $s_i \cdots s_{j-1}$ do not. A *parsing scheme* is a way of producing parsings for strings. We consider parsing schemes given by a pair of disjoint alphabets $\Sigma_0, \Sigma_1 \subseteq \Sigma$. This defines a parsing in the following way:

repetitions We group into a phrase each maximal *repetition* a^ℓ with $a \in \Sigma$ and $\ell > 1$.
pairs We group each $ab \in \Sigma_0\Sigma_1$ into a phrase.
All the remaining letters form length-1 phrases.

Next, we construct a new alphabet which has a letter for each constructed phrase.

Using the new alphabet, a parsing of w gives raise to a new string w', which is obtained by replacing phrases of length greater than 1 by their new symbols. This is called a *signature* of the string w and denoted by $\mathrm{sig}(w)$. Note that the signature depends on the parsing scheme (i.e. Σ_0 and Σ_1) as well as on the chosen symbols; the former is always clear from the context and the latter is ignored as the exact choice is irrelevant as long as it is consistent.

Given a letter a, its expansion $\exp(a)$ is the phrase that it replaced and its full expansion $\mathrm{Exp}(a)$ is the substring of the original text that it represents, which is obtained by iterative application of exp. This is generalised to strings in the obvious way.

Given two strings, we can find in linear time a parsing scheme which replaces those string with signatures that are shorter by a constant fraction.

▶ **Lemma 2.** *Given two strings s, t over an alphabet Σ we can find in time $\mathcal{O}(|s| + |t| + |\Sigma|)$ a parsing scheme of size at most $\frac{11}{12}|st| + \frac{1}{3}$ and produce the corresponding signatures.*

The proof is a variant of a known construction [11, 9]. The idea is that when Σ is randomly partitioned into Σ_0 and Σ_1, then among every two consecutive letters, with constant probability at least one is going to be parsed into a phrase of length two or more. Case inspection shows that the claim holds in expectation, and we can derandomise the procedure using the conditional expectations.

We call the parsing from Lemma 2 the *parsing for s, t*; given that there could be many such parsings, we choose one arbitrarily. We iterate the parsing process for two strings until they are reduced to single letters: an *iterated parsing scheme* is a sequence of parsing schemes $(\Sigma_{0,1}, \Sigma_{1,1}), (\Sigma_{0,2}, \Sigma_{1,2}), \ldots, (\Sigma_{0,\ell}, \Sigma_{1,\ell})$; its *height* is ℓ. Given a string s, an iterated parsing scheme defines a sequence of signatures $s = \mathrm{sig}_0(s), \mathrm{sig}_1(s), \mathrm{sig}_2(s), \ldots, \mathrm{sig}_\ell(s)$, in which $\mathrm{sig}_i(s)$ is the signature of $\mathrm{sig}_{i-1}(s)$ according to parsing scheme $(\Sigma_{0,i-1}, \Sigma_{1,i-1})$, where $\mathrm{sig}_{i-1}(s)$ is a string over $\Sigma_{i-1} = \Sigma_{0,i-1} \cup \Sigma_{1,i-1}$. Note that Σ_i and Σ_j for $i \neq j$ are not necessarily disjoint and in constructions they are usually not: not all letters from a string are replaced with their signatures, and so we want to replace them later on. We say that a letter a is from the ith level, or simply an i-letter, if $a \in \Sigma_i \setminus \bigcup_{j<i} \Sigma_j$.

▶ **Lemma 3.** *Given two strings s, t, there is an iterated parsing scheme of height $\mathcal{O}(\log |st|)$ such that $\mathrm{sig}_\ell(s)$ and $\mathrm{sig}_\ell(t)$ are letters.*

We call this parsing scheme the *parsing scheme for s, t*.

A parsing scheme is *locally consistent* if different occurrences of v (in the same or different strings) are parsed into the same phrases, possibly except $\mathcal{O}(1)$ beginning and ending phrases. Formally, if different occurrences of v are parsed into $s_1, \ldots, s_i$ and $s'_1, \ldots, s'_{i'}$, then there are $b, b', e, e' \in \mathcal{O}(1)$ such that the sequences of phrases $s_{1+b}, \ldots, s_{j-e}$ and $s'_{1+b'}, \ldots, s'_{j'-e'}$ are equal (in particular they have the same length).

▶ **Lemma 4.** *A parsing scheme defined by a partition of the alphabet are locally consistent.*

4 Approximation via embedding into normed vector spaces

Idea. While different occurrences of the same substring in s, t may be parsed differently by an iterated parsing scheme, the same symbol always fully expands to the same substring of the original strings s, t. This leads to a natural meta-algorithm for the common partition (and the edit distance with block moves) for s, t, which was first proposed by Cormode and Muthukrishnan [6] (earlier work [7] used a similar though more involved approach): given s, t calculate their iterated parsing scheme and set of signatures $s_0, \dots, s_k$ and $t_0, \dots, t_k$, where $k = \mathcal{O}(\log(st))$, and then iteratively look at i-symbols for $i = k, k-1, \dots$. If there are common symbols in s_i and t_i, then make corresponding full expansions in s, t as parts and match them to each other. For the remaining symbols, expand them to phrases in s_{i-1} and t_{i-1}. The algorithm depends only on the number of occurrences of each i-symbol in s_i an t_i; thus, we can represent s, t as vectors of counts of occurrences of letters in appropriate signatures. Amortised analysis shows that the size of the resulting partition (the number of edit moves) is within a constant factor of the ℓ_1 norm of the difference of the vectors for s and t. As a last step, one argues that this difference is at most $\mathcal{O}(k)$ times the size of the minimal common partition (the edit distance), which is shown by induction on the edit distance value. Adding insertion and deletion as allowed operations keeps the whole scheme more or less the same; in particular, we still use the ℓ_1 norm.

Unfortunately, allowing more operations (and in particular their combinations) distorts this approach. The needed modifications are explained at appropriate places.

Embedding to normed vector spaces. Given a string s and an iterated parsing scheme $(\Sigma_{0,i}, \Sigma_{1,i})_{i=1}^{k}$, let $s = s_0, \dots, s_k$ be the sequence of its signatures and let $\Sigma_i = \Sigma_{0,i} \cup \Sigma_{1,i}$. We embed s into a vector space whose coordinates are indexed with elements of $\bigcup_{i=0}^{k} \Sigma_i$: for an i-letter a, we set $V(s)[a] = |s_i|_a$, i.e. the number of occurrences of the letter a in the appropriate signature of s (the first one to use letter a). Define a symmetric difference $V(s) \triangle V(t)$ of such vectors as

$$(V(s) \triangle V(t))[a] = \lceil |V(s)[a] - V(t)[a]| \rceil \ .$$

Note that taking the ceiling is not needed, as coordinates are natural numbers, but it is used for vectors defined later on. We also define the *support* $\sup(v)$ of a vector, in which every non-zero component of v is replaced with 1, i.e. $\sup(v)[a] \in \{0, 1\}$ and $\sup(v)[a] = 0 \iff v[a] = 0$. Lastly, the standard ℓ_1 norm is the sum of its coordinates (which are all non-negative): $\|V(s)\|_1 = \sum_a V(s)[a]$. Define also $V_i(s)$ that restricts $V(s)$ to coordinates in $\bigcup_{j \le i} \Sigma_j$

Algorithms. We now give the algorithms for several variants of the edit distance with operations and bound their sizes in terms of vectors related to input strings. The basic case is when the move operation is allowed; it serves as a model for other algorithms.

Common partitions for repetitions. Our algorithms try to match identical symbols in two signatures, yet it is more beneficial to match long repetitions instead of single letters, i.e. partition repetitions into subrepetitions such that those of larger lengths can be matched using less parts. It turns out that this is a variant of the original common partition problem; we state the simple result for later reference.

▶ **Lemma 5.** *For two sets S, T of a repetitions with, respectively, n_S and n_T repetitions and having the same sum of lengths of repetitions, there exists a common partition between S and T of size at most $2n_S + 2n_T$.*

It is enough to match any repetition from one set to a prefix in the other.

Move. AlgMove works as follows: We compute the iterated parsing scheme for s, t, the corresponding signatures $s_0, \ldots, s_k$ and $t_0, \ldots, t_k$, and the vectors $V(s)$ and $V(t)$. In the same time bounds, we can also create the list of occurrences of a in s_i and t_i for each i-letter a. Also, for each letter in the signature, we compute the beginning and the end of the corresponding full expansions in the original string.

During the algorithm, we consider the strings s_i, t_i for $i = k, \ldots, 1, 0$. We colour some letters of s_i, t_i black, such that the multisets of black coloured letters in s_i, t_i are the same.

Initially, there are no coloured letters in s_k, t_k. For each i we proceed as follows: first, we consider s_i, t_i with the coloured letters removed, which yields two sets of strings, called S and T, respectively. For each i-letter a, take the sets of all maximal a-repetitions in S and T (which includes those of length 1, i.e. single letters a). Let their total sum of lengths be ℓ_s, ℓ_t, respectively, and let $\ell = \min(\ell_s, \ell_t)$. Choose among those two sets (sub)repetitions with total length ℓ (we take all repetitions from one of the sets, while in the other we may need to split one repetition). Let the numbers of the chosen repetitions be n_s, n_t, respectively. Using Lemma 5, we find a common partition for them of size at most $2(n_s + n_t)$. We then colour those letters black, remove them from S, T and declare their full expansions in s and t as parts and map the ones in s to t. If the removal happens in the middle of some string in $S \cup T$, then this string is split into two strings and both are added back to the appropriate set. After that, we expand each i-letter to the corresponding phrase in s_{i-1} or t_{i-1}; the expansion is black coloured if and only if this letter is black coloured.

After processing 0-letters, the final actions depend on the allowed operations: if there are any uncoloured letters in $s_0 = s$, then we delete them or reject if deletion is not allowed; similarly, if there are any uncoloured letters in $t_0 = t$, then we insert them or reject if insertion is not allowed.

▶ **Lemma 6** (cf. [6]). *Given an iterated parsing scheme for strings s and t,* AlgMove *constructs in linear time a common partition of size $\mathcal{O}(\|V(s) \triangle V(t)\|_1)$.*

Proof.

▶ **Claim.** *When the algorithm processes s_i, for each a the number of black coloured letters a in s_i and t_i is the same.*

This is true when there are no coloured letters; we show that this number changes in the same way for s and t. When we expand the letters, by the inductive assumption the multiset of black-coloured letters is the same in s_i and t_i. Each such letter is replaced with the same expansion, so the claim holds also after the expansion. When we colour letters, we do it on the same (multi)sets of letters in s_i, t_i.

Claim 4 implies that after processing an i-letter a, but before the expansion, the number of uncoloured a's in s_i, t_i is exactly $(V(s) \triangle V(t))[a]$: those uncoloured letters are exactly in one of s_i, t_i and the coloured letters have the same number of occurrences in s_i and t_i.

Concerning the cost, we assume that the creation of one part in the common partition consumes one unit of credit. We keep the invariant that right before processing i-letters, each repetition in S and T (including length-1 repetitions) has 2 units of credit. The credit is spent when the partition is formed: The common partition of repetitions costs on average 2 per paired repetition, which is paid by the credit on this repetition. After the processing, the unused credit on the repetitions of i-letters is discarded and 4 fresh units of credit are issued to each i-level symbol that has not been removed (i.e., coloured black). Recall that a fixed i-letter a has exactly $(V(s) \triangle V(t))[a]$ such occurrences, so in in total $4\|V(s) \triangle V(t)\|_1$ units of credit are issued.

If $i > 0$, this credit freshly assigned to an i-letter a is then reassigned to letters in the expansion of a: if $\exp(a)$ is a repetition, 4 units are reassigned to this repetition, if it is a pair, 2 units of credit is given to each of those letters.

In case of $i = 0$, we observe that each remaining symbol is a 0-letter and has 4 units of fresh credit, which can be used to pay for the final operations of delete and insert. ◀

The second step of the analysis is to show that $\|V(s) \triangle V(t)\|_1$ indeed upper bounds the edit distance (multiplied by $\mathcal{O}(\log n)$).

▶ **Lemma 7** (cf. [6]). *Let s, t be two strings and let $\{\mathsf{mv}\} \subseteq \mathsf{Op} \subseteq \{\mathsf{ins}, \mathsf{del}, \mathsf{mv}\}$. Fix an iterated parsing scheme of height k. Then $\|V(t) \triangle V(s)\|_1 = \mathcal{O}(d(k+1))$, where $d = \mathrm{ED}_{\mathsf{Op}}(s,t)$.*

Proof. As $\|\cdot \triangle \cdot\|_1$ satisfies the triangle inequality, it is enough to give the proof for $d = 1$.

Let $s = s_0, \ldots, s_k$ and $t = t_0, \ldots, t_k$ be the consecutive signatures for s, t according to the parsing scheme and $V_0(s), \ldots, V_k(s)$ and $V_0(t), \ldots, V_k(t)$ be the corresponding vectors. We first show by induction on k a stronger claim for move and then adapt it to other operations:

▶ **Claim.** *There are at most 3 substrings in s_k and at most 3 substrings in t_k, called* difference strings*, of total length ℓ_k, such that the multisets of substrings of $\mathrm{sig}_k(s)$ and $\mathrm{sig}_k(t)$ obtained after the removal of the difference strings are equal and $4\ell_k + \|V_k(s) \triangle V_k(t)\|_1 = \mathcal{O}(k+1)$.*

For the base of the induction, if $s = w_1 w_2 w_3 w_4$ is turned to $t = w_1 w_3 w_2 w_4$, let the difference substrings be length-2 substrings on the boundary between each w_i and subsequent w_j. We merge the chosen substrings if they overlap or are adjacent, which results in at most 3 such substrings in s_0 and t_0; their total length is at most $\ell_0 = 12$. Clearly, $(V_0(s) \triangle V_0(t))[a] = 0$ as no letters are removed nor added.

For the induction step, consider how s_k and t_k are parsed. Define the difference strings in s_{k+1} and t_{k+1} as those whose expansions are contained in the difference strings in s_k, t_k or form the $\mathcal{O}(1)$ phrases around the difference substrings that may be parsed differently; see Lemma 4. So the increase ℓ_{k+1} from ℓ_k is upper bounded by $\mathcal{O}(1)$, but it can also decrease if there are phrases in the difference strings that are longer than 1.

Consider the multisets of strings obtained from s_{k+1}, t_{k+1} after the removal of the difference strings. By the choice of the difference strings, their expansions were parsed in the same way (see Lemma 4), and thus those multisets are identical. Consider now $V_{k+1}(s) \triangle V_{k+1}(t)$ and new letters (compared to $V_k(s) \triangle V_k(t)$). Those are $(k+1)$-letters and they are in the difference strings of s_{k+1}, t_{k+1}. Either they are one of the $\mathcal{O}(1)$ letters that replaced phrases that were parsed differently or letters whose phrases consists of the letters in the difference strings for s_k, t_k. But each of the latter letters decreases ℓ_{k+1} when compared to ℓ_k by at least 1: difference strings for s_k, t_k did not include any $(k+1)$-letters and each such a letter corresponds to a phrase of length at least 2.

For del and ins, the difference strings on the 0-th level include the deleted (or inserted) letter and otherwise the proof is only simpler (as there are fewer substrings after the removal of the difference strings and they are in the same order). ◀

▶ **Theorem 8.** AlgMove *gives an $\mathcal{O}(\log n)$ approximation of common partition problem with a set of operations which is any subset of insert, delete. Its running time is linear assuming integer sorting runs in linear time. The same applies to the edit distance with set of operations that include block move and any subset of operations of insert, delete.*

Move and block delete. We now investigate the case in which we allow move as well as block delete operation. This makes the situation asymmetric with respect to s and t. The algorithm is almost the same as AlgMove, though the analysis becomes more involved.

The differences between AlgBdel and AlgMove are as follows: the first is the treatment of the remaining uncoloured letters in s_0 after processing level-0 letters: we delete each maximal string of such letters using block delete. The second is that we make the common partition for repetitions in a more clever way (though it is still a valid one for AlgMove): for a fixed letter a, consider the a-repetitions in S, T (recall that those are the sets of uncoloured a-repetitions in s_i, t_i, respectively); let them have lengths $M = \{m_i\}_{i \in I}$ and $N = \{n_i\}_{i \in J}$. We make the common partition for a-(sub)repetitions in a two-step process. First, we match the a-repetitions of length 1: Consider the 1's that are common in M and N; we colour the corresponding a's in S, T black, remove them from S, T and make their full expansions parts in common partition, and update N, M by removing the common 1's. Note, that now $M \cap N = \emptyset$: if there is a^ℓ in both of them, then this a^ℓ was expanded from the same letter in s_{i+1} and t_{i+1}. But this is not possible, as we colour all such letters black.

For the remaining a-repetitions in S, T, let ℓ_s be the total length of a-repetitions in S (i.e. $\ell_s = \sum_{p \in M} p$), let ℓ_t be the corresponding total length of a-repetitions in T, and let $\ell = \min(\ell_s, \ell_t)$. Choose a-repetitions in S with a total length ℓ, preferring the longer repetitions. We make the common partition between the chosen repetitions in S and the ones in T of length ℓ.

Concerning the analysis, it is clear that $\|V(s) \triangle V(t)\|_1$ cannot be used, as for $t = \epsilon$ it is useless; $\|V(t) \setminus V(s)\|_1$ is a natural candidate, but it is not subtle enough: consider $s = (ab)^\ell$ and $t = a^\ell$. It is clear that at least ℓ operations are needed to transform s to t, yet $\|V(t) \setminus V(s)\|_1 = \mathcal{O}(1)$. The problem is that several short a-repetitions from s are needed to form one long a-repetition in t. On the other hand, identical a-repetitions should be "for free": when $s = t$, then we should not impose any cost.

Motivated by those examples, we define a new cost function for s, t. It is somehow related to Wassersteiner ("earth mover") distance, but it is directed and applies to sets with different sums as well. Let us first define it on multisets of natural numbers: given two such multisets $\{x_i\}_{i \in I}$ and $\{y_i\}_{i \in J}$, we first exclude from those sets their common part and look for the smallest number of elements in $\{x_i\}_{i \in I}$ whose sum is at least the sum of $\{y_i\}_{i \in J}$; if $\{x_i\}_{i \in I}$ is not enough, we pad it with an arbitrary number of 1's. Formally, let $I' \subseteq I$ and $J' \subseteq J$ be such that $\{x_i\}_{i \in I'} = \{x_i\}_{i \in I} \setminus (\{x_i\}_{i \in I} \cap \{y_i\}_{i \in J})$ and $\{y_i\}_{i \in J'} = \{y_i\}_{i \in J} \setminus (\{x_i\}_{i \in I} \cap \{y_i\}_{i \in J})$, define $x = \sum_{i \in I'} x_i$, $y = \sum_{i \in J'} y_i$. Then the $\mathrm{SD}(\{x_i\}_{i \in I'}, \{y_i\}_{i \in J'})$ is defined as follows: if $x < y$, then it is $(y - x) + |I'|$; otherwise, it is the smallest m such that the sum of the largest m elements in $\{x_i\}_{i \in I'}$ is at least y. Lastly, we set $\mathrm{SD}(\{x_i\}_{i \in I}, \{y_i\}_{i \in J})$ as $\mathrm{SD}(\{x_i\}_{i \in I'}, \{y_i\}_{i \in J'})$.

▶ **Lemma 9.** SD *satisfies the directed triangle inequality.*

Then for the input strings s, t and a j-letter a, we define $\mathrm{SD}(s, t)[a]$ as $\mathrm{SD}(\{x_i\}_{i \in I}, \{y_i\}_{i \in J})$, where $\{x_i\}_{i \in I}, \{y_i\}_{i \in J}$ are the multisets of lengths of a-repetitions in s_j and t_j, respectively. Note that, unlike embedding to vectors, $\mathrm{SD}(s, t)$ cannot be computed for s and t separately; it is defined for a pair s, t.

The following two lemmata are the counterparts of Lemma 6 and Lemma 7 in case when block delete is allowed; their proofs are similar.

▶ **Lemma 10.** *Let* $\{\textsf{b-del}, \textsf{mv}\} \subseteq \textsf{Op} \subseteq \{\textsf{b-del}, \textsf{mv}, \textsf{del}, \textsf{ins}, \textsf{uncp}\}$. *Given an iterated parsing string for strings s and t,* AlgBdel *construct a partition of size* $\mathcal{O}(\|V(t) \setminus V(s)\|_1 + \|\mathrm{SD}(s,t)\|_1)$. *Moreover, it runs in linear time.*

▶ **Lemma 11.** *Let s, t be two strings and let $\{\mathsf{b\text{-}del}, \mathsf{mv}\} \subseteq \mathsf{Op} \subseteq \{\mathsf{mv}, \mathsf{b\text{-}del}, \mathsf{ins}, \mathsf{del}, \mathsf{uncp}\}$. Fix an iterated parsing scheme of height k. Then $\|V(t) \setminus V(s)\|_1$, $\|\mathrm{SD}(s,t)\|_1 \in \mathcal{O}(d(k+1))$, where $d = \mathrm{ED}_{\mathsf{Op}}(s,t)$.*

▶ **Theorem 12.** AlgBdel *is an $\mathcal{O}(\log n)$ approximation of edit distance for a set of operations that include block move, block delete, and any subset of block uncopy, insert, delete. Its running time is linear assuming integer sorting running time is linear.*

Move and copy. We now give an algorithm that deals with the scenario in which both block move and block copy are allowed. As a simple example, consider $s = a^n$ and $t = a^m$, where $m \geq n$; the easiest way to obtain t is to repeatedly "square" the string $\lceil \log(m/n) \rceil$ times.

Thus, if copy is allowed, we need also to take into the account the lengths of maximal repetitions. To model this in the analysis,[1] given an iterated parsing scheme, we define a vector $\mathrm{LMax}(s)$, indexed by letters of $\bigcup_i \Sigma_i$, so that $\mathrm{LMax}(s)[a]$ for an i-letter a is the logarithm of the longest a-repetition in s_i; we set $\mathrm{LMax}(s)[a] = 0$ if there is no such repetition.

As a second part of the intuition, we note that having copied a symbol, after some expansions we may realise that it would be better to perform moves instead. Imagine that an i-letter a occurs twice in t_i and we declare one occurrence to be a copy of the other. Later on, a is expanded to bc, and it turns out that in s_{i-1} there are two uncoloured copies of b and c. In this case, it is better to cancel the copying and move two b's and two c's into t_{i-1}.

AlgBcp proceeds similarly as AlgMove: for $i = k, k-1, \ldots, 0$, we consider s_i, t_i. We construct move and copy operations: the move operations are performed in the order in which AlgBcp constructs them, their sources are always in s_i and targets in t_i; we copy only within t_i and those operations are performed in the reverse order (compared to how the algorithm constructs them) after all the other operations. This should be intuitively clear: when in t_i we declare one occurrence of a substring t' to be a copy of another occurrence, then it may be that we still do not know how t' is constructed from the substrings of s.

The target of a copy operation is coloured grey and this colour is preserved by expansions. However, we may always change our mind and uncolour any grey substring. To simulate this, we split the target into (at most 3) shorter blocks, replacing the original copy operation by more such operations, and we cancel one of them. In fact, we uncolour only to make room for the target of a move operation, so the uncoloured symbols are immediately coloured black.

Let the multisets of uncoloured letters in s_i, t_i be S, T, respectively. For each i-letter a, we list all a-repetitions in S, T: let ℓ_s and ℓ_t be the total length of a-repetitions in S and T.

- If $\ell_s \geq \ell_t$, then we use Lemma 5 to make a common partition of repetitions from S of total length ℓ_t and all repetition in T, colour those letters black, remove them from S, T, and move their full expansions from s_0 to t_0. If there are repetitions of a left in s_i, then we look whether there are any grey a-repetitions in t_i and we proceed as in Lemma 10, but in the other direction: we first match single a-repetitions in S and single a-repetitions coloured grey (so in T). After this operation, it cannot be that a repetition a^k has an uncoloured occurrence in s_i and a grey one in t_i, as this would mean that the $(i+1)$-letter representing a^k had such occurrences in s_{i+1} and t_{i+1}, which is not possible. Then we take the longest grey a-repetitions, enough to make the common partition with the repetitions in S, or all grey a-repetitions, if there are not enough of them. We make a common partition for those repetitions, recolouring the matched grey letters black and move the corresponding full expansions from s to t.

[1] This can be also solved by ensuring that the symbol that replaced a^k is not grouped in the next $\log k$ phases of the iterated parsing scheme [15]; this moves the burden from the analysis to the algorithm.

- If $\ell_s < \ell_t$, then we choose repetitions in T of total length ℓ_s, including the longest repetition of T or, if it is longer than ℓ_s, including its prefix of length ℓ_s. We colour the corresponding letters black and then move their full expansions from s to t. Next, we make sure that the longest repetition in t_i is fully coloured (except the first letter if $\ell_s = 0$). For this, we iteratively copy its longest coloured prefix to its following part, colouring the latter grey, which doubles the length of the coloured prefix of this repetition; if the longest repetition is fully uncoloured (i.e., if $\ell_s = 0$), then we begin with copying its first letter to the second. Finally, if there is any other uncoloured (sub)repetition left in T, then we colour it grey and mark it as a copy of the prefix of the longest repetition.

After processing all 0-letters, we perform the final operations as in AlgMove: when insertion is allowed, we insert all remaining letters in t (or reject, when insertion is not allowed) and delete all remaining letters in s, (or reject, when deletion is not allowed).

As before, the analysis has two steps: on one hand we estimate the cost in terms of various functions based on $V(s), V(t)$ (see Lemma 13) and on the other we show that those functions are bounded by $\mathcal{O}(d(k+1))$, where k is the height of the parsing scheme for s, t. When the appropriate functions are known, the proofs follow similarly as in Lemma 6 and 7 (recall that for a vector v the $\sup(v)$ changes each v's non-zero component to 1).

▶ **Lemma 13.** *Let $\{\mathsf{cp}, \mathsf{mv}\} \subseteq \mathsf{Op} \subseteq \{\mathsf{cp}, \mathsf{mv}, \mathsf{ins}, \mathsf{del}\}$. Given an iterated parsing scheme for strings s and t,* AlgBcp *returns a sequence of $\mathcal{O}(\|V(s) \setminus V(t)\|_1 + \|\sup(V(t)) \setminus \sup(V(s))\|_1 + \|\mathrm{LMax}(t) \setminus \mathrm{LMax}(s)\|_1 + \|\mathrm{SD}(t,s)\|_1)$ operations from Op that transform s to t. Moreover, it runs in linear time.*

▶ **Lemma 14.** *Let s, t be two strings and let $\{\mathsf{cp}, \mathsf{mv}\} \subseteq \mathsf{Op} \subseteq \{\mathsf{cp}, \mathsf{mv}, \mathsf{ins}, \mathsf{del}\}$ with a fixed iterated parsing scheme of height k. Then $\|V(s) \setminus V(t)\|_1$, $\|\sup(V(t)) \setminus \sup(V(s))\|_1$, $\|\mathrm{LMax}(s) \triangle \mathrm{LMax}(t)\|_1$ and $\|\mathrm{SD}(t,s)\|_1$ are in $\mathcal{O}(d(k+1))$, where $d = \mathrm{ED}_{\mathsf{Op}}(s,t)$.*

▶ **Theorem 15.** AlgBcp *gives an $\mathcal{O}(\log n)$ approximation of the edit distance for operations that include block copy and move and any operations from: insert, delete. Its running time is linear assuming integer sorting runs in linear time.*

Copy and uncopy. We now investigate the case in which both copy and uncopy operations are allowed. Although the move can be simulated by them, we still use the move operation as it makes the description of the algorithm and the analysis more similar to those of previous algorithms. As previously, AlgBcpuncp can deal also with letter insertions and deletions.

AlgBcpuncp, as AlgBcp, colours the letters grey or black to represent that they are already dealt with. Initially all letters are uncoloured. When we expand a letter, its expansion gets coloured if and only if the letter was coloured. While we construct the sequences of all operations in parallel, we in fact perform first all uncopy operations, then all moves and lastly all copy operations. Uncopying is always done witin s_i, those operations are performed in order of their construction. In such a case, we colour the uncopied grey letters. We move elements from s_i to t_i and those operations are performed in the order in which the algorithm constructs them; we colour both the source and target letters of this operation black. We copy only within t_i and those operations are performed in the reverse order (compared to how the algorithms constructs them). Targets of the copy (uncopy) operation are coloured grey. Concerning other operations, insertion and deletions are done all at once, after all uncopy and move operations but before copy operations.

We compute the iterated parsing scheme and process the strings s_i, t_i in phases for $i = k, k-1, \ldots, 0$. In the ith phase, we consider each i-letter a and introduce some move, copy, and uncopy operations to make sure that if a occurs in both s_i and t_i, then all the occurrences are coloured; otherwise, exactly one occurrence shall be uncoloured. Let the lengths of the longest a-repetition in s_i and t_i be ℓ_s and ℓ_t, respectively (these values can be equal to 0 if a does not occur in s_i or t_i). Fix some occurrences of those longest a-repetitions, preferring black, then uncoloured, and then grey. We uncopy each uncoloured a-repetition in s_i (except the chosen one) from the chosen one and, symmetrically, copy each uncoloured a-repetition in t_i (except the chosen one) from the chosen one; the targets of those operations are coloured grey. Now the actions depends on whether those chosen repetitions are coloured. To streamline the argument for $\min(\ell_s, \ell_t) = 0$, we assume that an empty repetition is black.

- If they are both coloured, then we do nothing: all a-repetitions in both s_i, t_i are coloured.
- If they are both uncoloured, then we move $\min(\ell_s, \ell_t)$ letters a from s_i to t_i. If $\ell_s > \ell_t$, then using $\lceil \log(\ell_s/\ell_t) \rceil$ uncopy operations we colour the rest of the chosen repetition in s_i; if $\ell_t > \ell_s$, then, symmetrically, the rest of this longest a-repetition in t_i is coloured grey using $\lceil \log(\ell_t/\ell_s) \rceil$ copy operations.
- If the one in s_i is coloured and the one in t_i is not, then the one in s_i must be black by the choice of the longest repetition (grey last): it is impossible that all repetitions a^{ℓ_s} in s_i are grey. Furthermore, it can be shown that there is a black repetition a^{ℓ_s} in t_i. Since we chose uncoloured repetition in t_i, by the choice strategy (black first) it holds that $\ell_t > \ell_s$. If $\ell_s > 0$, we copy the chosen uncoloured repetition a^{ℓ_t} from the black repetition a^{ℓ_s} in t, using $1 + \lceil \log(\ell_t/\ell_s) \rceil$ copy operations. Otherwise, we leave the first character of a^{ℓ_t} uncoloured and colour the remaining letters grey using $\lceil \log \ell_t \rceil$ copy operations.
- If the one in t_i is coloured and the one in s_i is not, the algorithm is symmetric to the previous case.

▶ **Lemma 16** (cf. [15, 16]). *Let* $\{\mathsf{cp}, \mathsf{uncp}\} \subseteq \mathsf{Op} \subseteq \{\mathsf{cp}, \mathsf{uncp}, \mathsf{mv}, \mathsf{ins}, \mathsf{del}\}$. *Given an iterated parsing scheme for strings s and t, one can construct in linear time a sequence of* $\mathcal{O}(\|\mathrm{LMax}(s) \bigtriangleup \mathrm{LMax}(t)\|_1 + \|\sup(V(s)) \bigtriangleup \sup(V(t))\|_1)$ *operations from* Op *that transform s to t.*

The bound of $\mathcal{O}(d(k+1))$ on $\|\sup(V(t)) \bigtriangleup \sup(V(s))\|_1$ and $\|\mathrm{LMax}(t) \bigtriangleup \mathrm{LMax}(s)\|_1$ follows already from Lemmata 7, 11, and 14.

▶ **Theorem 17.** AlgBcpuncp *is an $\mathcal{O}(\log n)$ approximation of the edit distance with set of operations that include block copy and uncopy and any subset of insert, delete, block move. Its running time is linear assuming integer sorting runs in linear time.*

5 Compressed Input

A *Straight-Line Programme* (SLP) is a context-free grammar that produces exactly one string and is treated as a compressed representation of this string. Its size is the sum of lengths of the right-hand sides of the productions.

The presented algorithms can be also implemented, when the input (i.e. strings s, t) are given as SLPs. In such a case, the running time increases to $\mathcal{O}(n \log N)$ and the approximation ratio is $\mathcal{O}(\log N)$, where n is the size of the SLPs representing s, t in the input and $N = \max(|s|, |t|)$ is the maximum of the lengths of strings defined by those SLPs.

The algorithms require only an implementation of the iterated parsing scheme for strings given as SLPs, which is known; see for instance [9, 12]. This is no surprise, as such techniques were introduced and are developed mostly in the context of grammar-compressed data.

References

1 Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Pattern matching in dynamic texts. In David B. Shmoys, editor, *11th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2000*, pages 819–828. ACM/SIAM, 2000. URL: `http://dl.acm.org/citation.cfm?id=338219.338645`.

2 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 51–58. ACM, 2015. `doi:10.1145/2746539.2746612`.

3 Laurent Bulteau and Christian Komusiewicz. Minimum common string partition parameterized by partition size is fixed-parameter tractable. In Chandra Chekuri, editor, *25th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 102–121. SIAM, 2014. `doi:10.1137/1.9781611973402.8`.

4 Marek Chrobak, Petr Kolman, and Jirí Sgall. The greedy algorithm for the minimum common string partition problem. *ACM Transactions on Algorithms*, 1(2):350–366, 2005. `doi:10.1145/1103963.1103971`.

5 Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986. `doi:10.1016/S0019-9958(86)80023-7`.

6 Graham Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *ACM Transactions on Algorithms*, 3(1):2:1–2:19, 2007. `doi:10.1145/1219944.1219947`.

7 Graham Cormode, Mike Paterson, Süleyman Cenk Sahinalp, and Uzi Vishkin. Communication complexity of document exchange. In David B. Shmoys, editor, *11th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2000*, pages 197–206. ACM/SIAM, 2000. URL: `http://dl.acm.org/citation.cfm?id=338219.338252`.

8 Funda Ergün, S. Muthukrishnan, and Süleyman Cenk Sahinalp. Comparing sequences with segment rearrangements. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2003*, volume 2914 of *Lecture Notes in Computer Science*, pages 183–194. Springer, 2003. `doi:10.1007/978-3-540-24597-1_16`.

9 Paweł Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Łącki, and Piotr Sankowski. Optimal dynamic strings. In Artur Czumaj, editor, *29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1509–1528. SIAM, 2018. `doi:10.1137/1.9781611975031.99`.

10 Avraham Goldstein, Petr Kolman, and Jie Zheng. Minimum common string partition problem: Hardness and approximations. *Electronic Journal of Combinatorics*, 12, 2005. URL: `http://www.combinatorics.org/Volume_12/Abstracts/v12i1r50.html`.

11 Artur Jeż. Approximation of grammar-based compression via recompression. *Theoretical Computer Science*, 592:115–134, 2015. `doi:10.1016/j.tcs.2015.05.027`.

12 Artur Jeż. Faster fully compressed pattern matching by recompression. *ACM Transactions on Algorithms*, 11(3):20:1–20:43, 2015. `doi:10.1145/2631920`.

13 Daniel P. Lopresti and Andrew Tomkins. Block edit models for approximate string matching. *Theoretical Computer Science*, 181(1):159–179, 1997. `doi:10.1016/S0304-3975(96)00268-X`.

14 Kurt Mehlhorn, R. Sundar, and Christian Uhrig. Maintaining dynamic sequences under equality tests in polylogarithmic time. *Algorithmica*, 17(2):183–198, 1997. `doi:10.1007/BF02522825`.

15 S. Muthukrishnan and Süleyman Cenk Sahinalp. Approximate nearest neighbors and sequence comparison with block operations. In F. Frances Yao and Eugene M. Luks, editors,

32nd Annual ACM Symposium on Theory of Computing, STOC 2000, pages 416–424. ACM, 2000. doi:10.1145/335305.335353.

16 S. Muthukrishnan and Süleyman Cenk Sahinalp. Simple and practical sequence nearest neighbors with block operations. In Alberto Apostolico and Masayuki Takeda, editors, *Combinatorial Pattern Matching, CPM 2002*, volume 2373 of *LNCS*, pages 262–278. Springer, 2002. doi:10.1007/3-540-45452-7_22.

17 Süleyman Cenk Sahinalp and Uzi Vishkin. On a parallel-algorithms method for string matching problems. In Maurizio A. Bonuccelli, Pierluigi Crescenzi, and Rossella Petreschi, editors, *Algorithms and Complexity, CIAC 1994*, volume 778 of *LNCS*, pages 22–32. Springer, 1994. doi:10.1007/3-540-57811-0_3.

18 Süleyman Cenk Sahinalp and Uzi Vishkin. Symmetry breaking for suffix tree construction. In Frank Thomson Leighton and Michael T. Goodrich, editors, *26th Annual ACM Symposium on Theory of Computing, STOC 1994*, pages 300–309. ACM, 1994. doi:10.1145/195058.195164.

19 Dana Shapira and James A. Storer. Edit distance with move operations. *Journal of Discrete Algorithms*, 5(2):380–392, 2007. doi:10.1016/j.jda.2005.01.010.

20 Dana Shapira and James A. Storer. Edit distance with block deletions. *Algorithms*, 4(1):40–60, 2011. doi:10.3390/a4010040.

21 Esko Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64(1-3):100–118, 1985. doi:10.1016/S0019-9958(85)80046-2.

A QPTAS for Gapless MEC

Shilpa Garg
Max Planck Institute for Informatics, Saarland Informatics Campus, Germany
sgarg@mpi-inf.mpg.de
https://orcid.org/0000-0002-1825-0097

Tobias Mömke[1]
University of Bremen and Saarland University, Saarland Informatics Campus, Germany
moemke@cs.uni-saarland.de
https://orcid.org/0000-0002-2509-6972

Abstract

We consider the problem Minimum Error Correction (MEC). A MEC instance is an $n \times m$ matrix M with entries from $\{0, 1, -\}$. Feasible solutions are composed of two binary m-bit strings, together with an assignment of each row of M to one of the two strings. The objective is to minimize the number of mismatches (errors) where the row has a value that differs from the assigned solution string. The symbol "$-$" is a wildcard that matches both 0 and 1. A MEC instance is gapless, if in each row of M all binary entries are consecutive.

Gapless-MEC is a relevant problem in computational biology, and it is closely related to segmentation problems that were introduced by [Kleinberg–Papadimitriou–Raghavan STOC'98] in the context of data mining.

Without restrictions, it is known to be UG-hard to compute an $O(1)$-approximate solution to MEC. For both MEC and Gapless-MEC, the best polynomial time approximation algorithm has a logarithmic performance guarantee. We partially settle the approximation status of Gapless-MEC by providing a quasi-polynomial time approximation scheme (QPTAS). Additionally, for the relevant case where the binary part of a row is not contained in the binary part of another row, we provide a polynomial time approximation scheme (PTAS).

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis, Theory of computation → Dynamic programming

Keywords and phrases approximation algorithms, QPTAS, minimum error correction, segmentation, computational biology

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.34

Acknowledgements We would like to thank Tobias Marschall for helpful discussions.

1 Introduction

The minimum error correction problem (MEC) is a segmentation problem where we have to partition a set of length m strings into two classes. A MEC instance is given by a set of n strings over $\{0, 1, -\}$ of length m, where the symbol "$-$" is a wildcard symbol. The strings are represented by an $n \times m$ matrix M, where the ith string determines the ith row $M_{i,*}$ of M. The distance dist of two symbols a, a' from $\{0, 1, -\}$ is $\text{dist}(a, a') := 1$ if $a = 0, a' = 1$ or $a = 1, a' = 0$ and $\text{dist}(a, a') := 0$ otherwise.

For two strings s, s' from $\{0, 1, -\}^m$ where s_j, s'_j denotes the j-th symbol of the respective string, $\text{dist}(s, s') := \sum_{j=1}^{m} \text{dist}(s_j, s'_j)$. A feasible solution to MEC is a pair of two strings

[1] Deutsche Forschungsgemeinschaft grant MO2889/1-1

© Shilpa Garg and Tobias Mömke;
licensed under Creative Commons License CC-BY

σ, σ' from $\{0,1\}^m$. The optimization goal is to find a feasible solution (σ, σ') that minimizes $\text{cost}_M(\sigma, \sigma') := \sum_{i=1}^{n} \min\{\text{dist}(M_{i,*}, \sigma), \text{dist}(M_{i,*}, \sigma')\}$. If M is clear from the context, we sometimes skip the index.

A MEC instance is called *gapless* if in each of the n rows of M, all entries from $\{0,1\}$ are consecutive. (As regular expression, a valid row is a word of length m from the language $-^*\{0,1\}^*-^*$). The MEC problem restricted to gapless instances is Gapless-MEC.

Our motivation to study Gapless-MEC stems from its applications in computational biology. Humans are diploid, and hence there exist two versions of each chromosome. Determining the DNA sequences of these two chromosomal copies – called haplotypes – is important for many applications ranging from population history to clinical questions [17, 18]. Many important biological phenomena such as compound heterozygosity, allele-specific events like DNA methylation or gene expression can only be studied when haplotype-resolved genomes are available [11].

Existing sequencing technologies cannot read a chromosome from start to end, but instead deliver small pieces of the sequences (called reads). Like in a jigsaw puzzle, the underlying genome sequences are reconstructed from the reads by finding the overlaps between them.

The upcoming next-generation sequencing technologies (e.g., Pacific Biosciences) have made the production of relatively long contiguous sequences with sequencing errors feasible, where the sequences come from both copies of chromosome. These sequences are aligned to a reference genome or to a structure called contig. We can formulate the result of this process as a Gapless-MEC instance: the sequences are the contiguous strings and the contig determines the columns of the strings.

Gapless-MEC is a generalization of a problem called Binary-MEC, the version of MEC with only instances M where all entries of M are in $\{0,1\}$. Finding an optimal solution to Binary-MEC is equivalent to solving the hypercube 2-segmentation problem (H2S) which was introduced by Kleinberg, Papadimitriou, and Raghavan [9, 10] and which is known to be NP-hard [4, 10]. The optimization version of Binary-MEC differs from H2S in that we minimize the number of mismatches instead of maximizing the number of matches. Binary-MEC allows for good approximations. Ostravsky and Rabiny [13] obtained a PTAS for Binary-MEC based on random embeddings. Building on the work of Li et al. [12], Jiao et al. [8] presented a deterministic PTAS for Binary-MEC.

Gapless-MEC was shown to be NP-hard by Cilibrasi et al. [3].[2] Additionally, they showed that allowing a single gap in each string renders the problem APX-hard. More recently, Bonizzoni et al. [2] showed that it is unique games hard to approximate MEC with constant performance guarantee, whereas it is approximable within a logarithmic factor in the size of the input. To our knowledge, previous to our result their logarithmic factor approximation was also the best known approximation algorithm for Gapless-MEC.

1.1 Our results

Our main result is the following theorem.

▶ **Theorem 1.** *There is a QPTAS for* Gapless-MEC.

Thus we partially settle the approximability for this problem: Gapless-MEC is not APX-hard unless NP $\subseteq$ QP (cf. [16]). Thus our result reveals a separation of the hardness of the gapless case and the case where we allow a single gap. Furthermore, already Binary-MEC is

[2] Their result predates the hardness result of Feige [4] for H2S. The proof of the claimed NP-hardness of H2S by Kleinberg, Papadimitriou, and Raghavan [9] was never published.

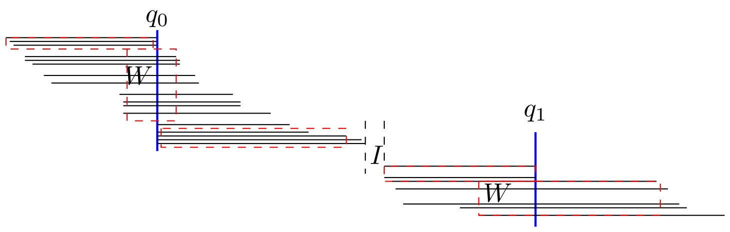


Figure 1 Subinterval-free instance. Blocks represented by ranges shown in red on an instance M and the blue lines are the columns, I and W shows the empty interval and central region respectively.

strongly NP-hard since the input does not contain numerical values. Therefore we can exclude the existence of an FPTAS for both BINARY-MEC and GAPLESS-MEC unless P = NP.

Additionally, we address the class of *subinterval-free* GAPLESS-MEC instances where no string is contained in another string. More precisely, for each pair of rows from M we exclude that the set of columns with binary entries from one row is a strict subset of the set of columns with binary entries from the other row.

▶ **Theorem 2.** *There is a PTAS for* GAPLESS-MEC *restricted to instances such that no string is the substring of another string.*

1.2 Overview of our approach

Our algorithm is a dynamic program (DP) that is composed of several levels. Given a general GAPLESS-MEC instance, we decompose the rows of the instance into length classes according to the length of the contiguous binary parts of the rows. For each length class we consider a well-selected set of columns such that each row crosses at least one column and at most two. (Row i crosses a column j, if $M_{i,j} \in \{0, 1\}$.)

We decompose each length class into two sub-classes, one that crosses exactly one column and one that crosses exactly two columns. For the second class, it is sufficient to consider every other column, which leaves us with many *rooted* instances. Thus for each sub-instance there is a single column (the root) which is crossed by all rows of the instance.

We further decompose rooted sub-instances into the left hand side and the right hand side of the root. Since the two sides are symmetric, we can arrange the rows and columns of these sub-instances in such a way that all rows cross the first column. We call this type of sub-instance *SWC-instance* (for "simple wildcards"). We order the rows from top to bottom by increasing length in order to be able to further decompose the instance.

The first level of our DP solves these highly structured SWC-instances. The basic idea that we would like to apply is that we select a constant number of rows from the instance that represents the solution. Without further precautions, however, this strategy fails because of differing densities within the instance: the selected rows have to represent both the entries of columns crossed by many short rows and entries of arbitrarily small numbers of rows crossing many columns. To resolve this issue, we observe that computing the solution strings σ and σ' is equivalent to finding a partition of M into two row sets, one assigned to σ and the other assigned to σ'. If we assume to have the guarantee that for both solution strings σ and σ' an ε fraction of rows of the matrix M forms a BINARY-MEC sub-instance, we show that the basic idea works.

This insight motivates to separate SWC-instances from left to right into sub-instances with the required property and to assemble them from left to right using a DP. There are, however, several complications. In order to choose the right sub-instances, we have to take into account that the choice depends on which rows are assigned to σ and which are assigned to σ'. Therefore the DP has to take special care when identifying the sub-instances.

Furthermore, in order to stitch sub-instances together to form a common solution, the solution computed in the left sub-instance has to compute a set of candidate solutions oblivious of the choices of the right sub-instance. This means that we have to compute a solution to the left sub-instance without looking at a fraction of rows. We present an algorithm for these sub-instances in Section 2.

In order to combine the sub-instances, we face further technical complications due to having distinct sub-instances for those rows assigned to σ and those rows assigned to σ'. In Section 2.1, we introduce a DP whose DP cells are pairs of simpler DP cells, one for σ and one for σ'.

Before we consider general instances, in Section 3 develop our techniques by considering subinterval-free instances which are easier to handle (see Fig. 1). Observe that the instances considered until now are special rooted sub-interval-free instances. We show how to solve arbitrary rooted sub-interval-free instances by combining the DP with additional information about the sub-problems that contain the root. We then introduce the notion of domination in order to combine rooted sub-interval-free instances with a DP proceeding from left to right. The main idea is that a dominant sub-problem dictates the solution. At the interface of two sub-instances, there can be a (contiguous) region where none of the two sub-problems is dominant. We show that these regions can be solved directly by considering a constant number of rows (using the results from Section 2).

Until this point, all parts of our algorithm run in polynomial time. We lose this property when considering length classes, in Section 4.1. The length classes allow us to separate an instance into rooted sub-instances. The difficulty is that the left hand side of a separating column may have a completely different structure than the right hand side of that column. We do not know how to combine the two sides by considering only a polynomial number of possibilities. If we allow, however, quasipolynomial running time, we can solve the problem. We use that each of the two sub-instances (the one on the left and the one on the right) is composed of at most logarithmically many parts. Considering all parts simultaneously allows us to take care of dependencies between the left hand side and the right hand side and still solve them as if they were separate instances. Combining such rooted instances from left to right then can be done in the same spirit as combining rooted sub-interval-free instances. To solve the entire length-class, we combine both solutions by running a new DP that considers quadruples of DP cells.

Finally, in Section 4.2, we are able to handle all length classes simultaneously. We solve general instances in the same spirit as the combined sub-instances of a single length class. Instead of considering quadruples of cells, however, we form collections of quadruples that are – figuratively speaking – stacked on top of each other. The key insight is that there are only $O(\log(n))$ different length classes and each collection has at most one quadruple of each length class. Considering all possible collections adds another power of $\log(n)$ to the running time, which is still quasi-polynomial.

1.3 Further related work

Binary-MEC is a variant of the Hamming k-Median Clustering Problem when $k = 2$ and there are PTASs known [8, 13]. Li, Ma, and Wang [12] provided a PTAS for the general consensus pattern problem which is closely related to MEC. Additionally, they provided a PTAS for a restricted version of the star alignment problem aligning with at most a constant number of gaps in each sequence.

Alon and Sudakov [1] provided a PTAS for H2S, the maximization version of Binary-MEC and Wulff, Urner and Ben-David [19] showed that there is also a PTAS for the maximization version of MEC. For MEC, He et al. [7] studied the fixed-parameter tractability in the parameter of fragment length with some restrictions. These restrictions allow their dynamic programming algorithm to focus on the reconstruction of a single haplotype and, hence, to limit the possible combinations for each column. There is an FPT algorithm parameterized by the coverage [14, 6]. Bonizzoni et al. [2] provided FPT algorithms parameterized by the fragment length and the total number of corrections for MEC. There are some tools which can be used in practice to solve MEC instances [15, 14].

Most research in haplotype phasing deals with exact and heuristic approaches to solve MEC. Exact approaches, which solve the problem optimally, include integer linear programming [5] and fixed-parameter tractable algorithms [7, 15].

1.4 Preliminaries and notation

We consider a Gapless-MEC instance, which is a matrix $M \in \{0, 1, -\}^{n\times m}$. The ith row of M is the vector $M_{i,*} \in \{0, 1, -\}^{1\times m}$ and the jth column is the vector $M_{*,j} \in \{0, 1, -\}^{n\times 1}$. The length of the binary part in $M_{i,*}$ is $|M_{i,*}|$. We say that the ith row of M *crosses* the jth column if $M_{i,j} \in \{0, 1\}$.

For each feasible solution (σ, σ') for M, we specify an assignment of rows $M_{i,*}$ to solution strings. The default assignment is specified as follows. For a row $M_{i,*}$, we assign $M_{i,*}$ to σ if $\text{dist}(\sigma, M_{i,*}) \leq \text{dist}(\sigma', M_{i,*})$. Otherwise we assign $M_{i,*}$ to σ'. For the rows of M assigned to σ we write $\sigma(M)$ and for the rows assigned to σ' we write $\sigma'(M)$. For a given instance, $\mathsf{Opt} = (\tau, \tau')$ denotes an optimal solution. Observe that knowing Opt allows us to obtain an optimal assignments $\tau(M)$ and $\tau'(M)$ by assigning each row to the solution string with fewest errors and knowing $\tau(M)$ and $\tau'(M)$ allows us to obtain an optimal solution by selecting the column-wise majority values.

2 Simple instances with wildcards

We consider instances of Gapless-MEC where all entries of column one in M are zero or one, i.e., $M_{i,1} \in \{0, 1\}$ for each index i. Observe that the wildcards now have a simple structure which we refer to as SWC-structure. An instance with SWC-structure is an SWC-instance.

▶ **Definition 3** (Standard ordering of SWC-instances)**.** We define the *standard ordering* of rows in M such that $|M_{i,*}| \leq |M_{i+1,*}|$ for each i, i.e., we order them from top to bottom in increasing length of the binary part.

▶ **Definition 4** (Good SWC-instances)**.** We call an SWC-instance M *good*, if it is in standard ordering and there are at least $\varepsilon|\tau(M)|$ rows of $\tau(M)$ and at least $\varepsilon|\tau'(M)|$ rows of $\tau'(M)$ that have only entries from $\{0, 1\}$.

To solve good SWC-instances, we generalize the PTAS for Binary-MEC by Jiao et al. [8]. Our algorithm requires partitions of the set of rows. In the following two definitions, the required number of rows may be a fractional number. To solve the problem, we allow the assignment of fractional rows, i.e., for a row i, we can choose an $x \in [0, 1]$ and assign an x fraction of i to one set and a $1 - x$ fraction to the other set.

The following two definitions allow us to introduce a structured view on optimal solutions.

```
Algorithm 1: SWC$_\delta$.
Input  : Row sets $U_i$, $L_i$, $U'_i$ and $L'_i$ of a good SWC-instance $M$, numbers $r, r'$.
         Optional: selection of rows $\tilde{U}_i, \tilde{L}_i, \tilde{U}'_i, \tilde{L}'_i$, see below.
Output : A pair of solution strings $(\sigma, \sigma')$.
Run the algorithm for each possible selection of the following type and keep the best
  outcome (minimum number of errors);        // If provided as input, skip
  selection.
For each $i$, select (with repetition) a multi-set $\tilde{U}_i$ of $1/\delta$ rows from $U_i$ and $\tilde{L}_i$ from $L_i$;
For each $i$, select (with repetition) a multi-set $\tilde{U}'_i$ of $1/\delta$ rows from $U'_i$ and $\tilde{L}'_i$ from $L'_i$
  such that $\tilde{U}' \cap \tilde{U} = \tilde{L}' \cap \tilde{L} = \emptyset$;
// $\tilde{U} := \bigcup_i \tilde{U}_i$. The values $\tilde{U}'$, $\tilde{L}$, and $\tilde{L}'$ are defined analogously.
For each column $j$, set $\sigma_j := \textsc{Majority}_j(\tilde{U}, \tilde{L})$ and $\sigma'_j := \textsc{Majority}_j(\tilde{U}', \tilde{L}')$;
For each row $i$ of $M$, determine the value $d_i := \mathrm{dist}(\sigma, M_{i,*}) - \mathrm{dist}(\sigma', M_{i,*})$;
Assign the $r$ rows with minimal values $d_i$ to $\sigma$ and the remaining $r'$ rows to $\sigma'$.
```

▶ **Definition 5** (Trisection). An *ε-trisection* of an instance M for τ is a partition of the rows into three consecutive ranges that have the following properties.

1. The first range U contains row $M_{1,*}$ and $(1-\varepsilon)|\tau(M)|$ rows of $\tau(M)$.
2. The second range L is consecutive to first row set containing $(\varepsilon - \varepsilon^2)|\tau(M)|$ rows of $\tau(M)$.
3. The third range X contains the remaining rows in M.

To avoid ambiguity, we choose L and X such that the first row is in $\tau(M)$.

We define an ε-trisection U', L', and X' for τ' analogously, replacing $\tau(M)$ by $\tau'(M)$.

▶ **Definition 6** (Subdivision of trisections). We consider the rows sets U, L, U', L' from Definition 5 and additionally, we divide each of these sets into $1/\varepsilon^2$ disjoint subsets denoted as U_i, L_i, U'_i, L'_i. For each i, U_i contains $\varepsilon^2 \cdot |U|$ rows from $\tau(M)$ and L_i contains $\varepsilon^2 \cdot |L|$ rows from $\tau(M)$. Analogously, each U'_i contains $\varepsilon^2 \cdot |U'|$ rows from $\tau'(M)$ and L'_i contains $\varepsilon^2 \cdot |L'|$ rows from $\tau'(M)$. To avoid ambiguity, each set U_i and L_i starts with a (fractional) row of $\tau(M)$ and each set U'_i and L'_i starts with a (fractional) row of $\tau'(M)$.

We introduce a new algorithm SWC$_\delta$ for our setting. For an instance M, we consider the rows sets U, L, U', L' from the ε-trisections of M and their subsets according to Definition 6. Additionally, we select a multi-set of rows from $U'_i \cap \tau'(M)$ and $L'_i \cap \tau'(M)$. We then compute the majority weighting according to Definition 7 for each column j using multisets based on the minimum number of errors. The main idea is to find two small row sets that represent the whole instance M. The intuitive meaning is that we select rows from the upper part with a much lower density then the rows of the lower part. We therefore introduce a bias such that all rows are equally important.

▶ **Definition 7** (Weighted majority). Let j be an integer and let $\tilde{U}$ and $\tilde{L}$ be two matrices with at least j columns. In $\tilde{U}_{*,j}$ and $\tilde{L}_{*,j}$, we replace all zeros by -1 and then all wildcard symbols by zero. We then compute the number $\nu := \sum_{i' \in \tilde{U}_{i,j}} (1-\varepsilon)i'/(\varepsilon - \varepsilon^2) + \sum_{i' \in \tilde{L}_{i,j}} i'$. Then $\textsc{Majority}_j(\tilde{U}, \tilde{L}) = 0$ if $\nu < 0$ and $\textsc{Majority}_j(\tilde{U}, \tilde{L}) = 1$ if $\nu \geq 0$.

With this preparation, we are now ready to present the algorithm. The input has a long list of parameters that will allow our dynamic programs later on to control the execution. The reason is that we do not know τ and τ'. Therefore the algorithm takes *guesses* of row sets as input. The values r and r' are guesses of $|\tau(M)|$ and $|\tau'(M)|$.

Observe that for small (i.e., constant) values of r or r', the algorithm SWC_δ can be replaced by an exact algorithm since we know $\tau(M)$ if and only if we know $\tau'(M)$, and we are able to guess constantly many rows.

▶ **Lemma 8.** *Let M be a good SWC-instance. For sufficiently large $r = |\tau(M)|$ and $r' = |\tau'(M)|$, let U_i, L_i, U'_i, L'_i be a subdivision (Definition 6) of an ε-trisection U, L, X, U', L', X' of M. Then $\mathrm{SWC}_{\varepsilon^3}$ is a $(1 + O(\varepsilon))$-approximation algorithm for M.*

The proof is based on a randomized argument using Chernoff bounds. In Lemma 8, we cannot control which rows of X and X' are assigned to which solution string. For our dynamic programs, we need a stronger statement. We would like to be able to compute a solution for an instance and *afterwards* change a fraction of assignments (guessing candidates for $\tau(X), \tau'(X')$) without losing the approximation guarantee. The next lemma is a key ingredient of our result.

▶ **Lemma 9.** *Let M be a good SWC-instance and $\varepsilon > 0$ sufficiently small. Let U, L, X be an ε-trisection for $\tau(M)$ and U', L', X' an ε-trisection for τ', with subdivisions U_i, L_i, U'_i, L'_i according to Definition 6. Let (σ, σ') be the solution computed by $\mathrm{SWC}_{\varepsilon^3}$ with $r = |\tau(M)|$, $r' = |\tau'(M)|$. Then re-assigning the rows $\sigma(X)$ to $\tau(X)$ and $\sigma'(X')$ to $\tau'(X')$ gives a $(1 + O(\varepsilon))$-approximation for the instance M.*

Proof. For ease of presentation, we assume that all appearing numbers are integers. It is easy to adapt the proof by rounding fractional numbers appropriately.

We first analyze the computed solution string σ. Let η be the total number of errors of (τ, τ') within M and let η_P be the total number of errors of (σ, σ') within $P := U \cup L$. Due to Lemma 8, we have $\eta_P \le (1 + O(\varepsilon))\eta$.

We may assume $r \ge r'$ since otherwise we can simply rename the two strings τ, τ'. Additionally, by renaming of σ and σ', we may assume that $|\sigma(P) \cap \tau(P)| \ge |\sigma'(P) \cap \tau(P)|$. Therefore $|\tau(P)| \ge n/3$ and $|\sigma(P) \cap \tau(P)| \ge n/6$. (Recall that the matrix M has n rows and m columns. The value $n/3$ is a safe bound on $n/2 - \varepsilon^2 n$, for $\varepsilon^2 \le 1/6$.)

▶ Claim 1. There is a set I of $m - 25\eta/n$ indices j such that $\sigma_j = \tau_j$ for all $j \in I$.

Proof of Claim. We concentrate on the columns of M where both strings τ and σ have at most $n/12$ errors within P. By counting the errors, there are at most $12\eta/n$ columns where τ has at least $n/12$ errors. Similarly, there are at most $12(1 + O(\varepsilon))\eta_P/n < 13\eta/n$ many columns where σ has at least $n/12$ errors. Therefore there is a set I of at least $m - 25\eta/n$ columns where simultaneously both τ and σ have less than $n/12$ errors each.

Now suppose that the claim was not true and there was an index $j \in I$ with $\tau_j \neq \sigma_j$. Then, since $|\tau(P) \cap \sigma(P)| \ge n/6$, either σ_j or τ_j is erroneous in at least $n/12$ rows of $\tau(P) \cap \sigma(P)$, a contradiction. ◊

Next we analyze σ' for the columns I. Let j be a column (i.e., an index) from I. By symmetry, we may assume $\sigma_j = \tau_j = 0$. We aim to show that an optimal solution has always sufficiently many errors to pay for wrong entries of σ'.

Let η_j be the number of errors of (τ, τ') in column j of M and let $\eta_{P,j}$ be the number of errors of (σ, σ') in column j of P. Let $\eta''_j = \eta_j + \eta_{P,j}$.

▶ Claim 2. For each column j of I, either $\sigma'_j = \tau'_j$ or $\eta''_j \ge (\varepsilon - \varepsilon^2)|\tau'(M)|/2$.

Proof of Claim. We distinguish two cases. We first assume $\tau'_j = 0$. If also $\sigma'_j = 0$, we are done. We therefore assume $\sigma'_j = 1$. If there are more than $|\tau'(L')|/2$ ones in column j of L', (τ, τ') has more than $|\tau'(L')|/2$ errors in column j and thus $\eta_j \ge |\tau'(L')|/2$. Otherwise

$\sigma'(L')$ has at least $|\tau'(L')|/2$ zeros in column j and therefore $\eta_{P,j} \geq |\tau'(L')|/2$. We obtain $\eta''_j \geq |\tau'(L')|/2 \geq (\varepsilon - \varepsilon^2)|\tau'(M)|/2$ as claimed.

In the second case, $\tau'_j = 1$ and we assume that $\sigma'_j = 0$. If there are more than $r'/2$ ones in column j of U', (σ, σ') has more than $r'/2$ errors in column j and thus $\eta_{P,j} \geq |\tau'(U')|/2$. Otherwise $\tau'(U')$ has at least $r'/2$ zeros in column j and therefore $\eta_j \geq |\tau'(U')|/2$. Again, we obtain $\eta''_j \geq |\tau'(U')|/2 \geq (1 - \varepsilon)|\tau'(M)|/2$ as claimed. ◊

Since by our assumption $|\tau'(X')| < \varepsilon^2|\tau'(M)|$, Claim 2 implies that within I, after reassigning the rows we still have a $(1 + O(\varepsilon))$-approximation.

To finish the proof, we argue that η is large enough to pay for all errors in X and X' outside of I. Let η_I be the number of errors due to assigning σ to $\tau(X)$ and σ' to $\tau'(X')$ within the interval I. Then, using the size of I stated in Claim 1, the total number of errors of (σ, σ') in M is at most $(1 + O(\varepsilon))\eta + \eta_I + \varepsilon^2 n \cdot 25\eta/n$, i.e., the errors of $\mathrm{SWC}_{\varepsilon^3}$ within P, the errors within X and X' in the columns of I, and all other entries of $X \cup X'$. The obtained approximation ratio is $((1+O(\varepsilon))\eta + \eta_I + \varepsilon^2 n \cdot 25\eta/n/\eta \leq (\eta + O(\varepsilon)\eta + 25\varepsilon^2\eta)/\eta = 1 + O(\varepsilon)$. The first inequality uses that for some constant k, $(1 + k\varepsilon)\eta \geq \eta + \eta_I$. ◀

2.1 A DP for SWC-instances

Let M be an SWC-instance with rows $\{1, 2, \dots n\}$. We define start_i to be the start and end_i the end of string number i of M, i.e., the column number of the matrix where the binary part starts and ends. For a sub-matrix M' of M, $\mathrm{start}_{M'}$ determines the index of the first column of M' and $\mathrm{end}_{M'}$ the index of the last column of M'. We next specify the parts of which the DP cells are composed. We divide the input instance into blocks defined as follows.

▶ **Definition 10** (Block). Given a good SWC-instance M, a block B is a sub-instance determined by three numbers $1 \leq a < b < c \leq n$ as follows. The first column of B is column 1 of M. The last column of B is end_b. The first row of B is a and the last row is n. We write U_B for the rows from a to $b - 1$, L_B for the rows from b to $c - 1$, and X_B for the rows from c to n.

The idea is that a block determines a trisection. We subdivide each block into chunks and select rows from these chunks. Chunks are closely related to subdivisions of trisections, but we do not assume the knowledge of (τ, τ').

▶ **Definition 11** (Chunk). Let B be a block determined by the numbers a, b, c. We partition B into $2/\varepsilon^2$ many *chunks* (ranges or rows). These chunks are determined by numbers $a = a_1 < a_2 < \cdots < a_{1/\varepsilon^2+1} = b = b_1 < b_2 < \cdots < b_{1/\varepsilon^2+1} = c$. The ℓth chunk of U_B is the submatrix composed of the rows a_ℓ to $a_{\ell+1} - 1$ and the ℓth chunk of L_B is the submatrix composed of the rows b_ℓ to $b_{\ell+1} - 1$.

▶ **Definition 12** (Selection). For each block B with a set of chunks C, we consider multiset T of rows of size $2/\varepsilon^5$. We require that T contains $1/\varepsilon^3$ rows from each chunk in C.

The selection T will take the role of $\tilde{U}$ and $\tilde{L}$ in SWC_δ.

▶ **Definition 13** (DP cell). For each block B, each set of chunks C of B and each selection T of rows from B, there is a DP cell represented by $D(B, C, T)$. A DP cell $D(B, C, T)$ is a *predecessor* of $D(\hat{B}, \hat{C}, \hat{T})$ if the following conditions hold.

- $\hat{a} = b$ and $\hat{b} = c$, where $b, c, \hat{a}, \hat{b}$ are the numbers from Definition 10.
- The chunks from C between b and c are exactly the chunks from $\hat{C}$ between $\hat{a}$ to $\hat{b}$.

- For each pair of chunks from $T \times \hat{T}$ with the same range of rows, the selections T and $\hat{T}$ restricted to the pair are the same.

The value of $D(B, C, T)$ will be an approximation of the minimum number of errors that we can have in M until the last column of B.

We now describe the dynamic program for a pair of solution strings (σ, σ') by using joint DP cells (ζ, ζ'). For σ', we use the same notation as in Definitions 10, 11 and 12, but we use the symbol prime ($\cdot'$) for all occurring variables.

▶ **Definition 14** (DP cell for a pair)**.** A joint DP cell $(\zeta, \zeta') = (D(B, C, T),\ D'(B', C', T'))$ is composed of two single cells defined as in Definition 13. We require that

- the rows of C and C' where chunks start are pairwise distinct, and
- $T \cap T' = \emptyset$.

▶ **Definition 15** (Predecessor of a joint DP cell)**.** A DP cell $(\hat{\zeta}, \hat{\zeta}')$ is a *predecessor* of (ζ, ζ') if (i) $\hat{\zeta} = \zeta$ and $\hat{\zeta}'$ is a predecessor of ζ'; or (ii) $\hat{\zeta}$ is a predecessor of ζ and $\hat{\zeta}' = \zeta'$.

Algorithm (SWC$^{\sigma,\sigma'}$). The general idea of the algorithm is to guess trisections. Suppose we initially chose blocks B, B' that are the trisections of the entire matrix M for τ and τ'. Then we obtain an approximation of the prefix of (τ, τ') restricted to B, B' (whichever ends first) by sampling rows of U_B,L_B,$U_{B'}$, and $L_{B'}$. The sampled rows for L_B and $L_{B'}$ provide the interface to the next step. Suppose $L_{B'}$ starts at an earlier row than L_B. Then we guess the trisection of M for τ restricted to the rows of $L_{B'}$ and $X_{B'}$. Let B'' be that block of our algorithm. Then $U_{B''} = L_B$ and we sample rows of $L_{B''}$ in order to approximate a new infix of τ. More precisely, the DP does the following.

We globally guess a number r that represent $|\tau(M)|$. Thus $r' := n - r$ represents $|\tau'(M)|$. We split the processing into an initialization phase and an update phase. In the initialization phase, we assign values to each DP cell (ζ, ζ') based on SWC$_{\varepsilon^3}$ with the following parameters. We obtain U_i, L_i from the chunks C and U_i', L_i' from the chunks C'. In the execution of SWC$_{\varepsilon^3}$, we use the selections T, T' instead of trying all possible selections, i.e., T and T' determine all $\tilde{U}_i$, $\tilde{L}_i$, $\tilde{U}_i'$, and $\tilde{L}_i'$ in the algorithm. Let $\tilde{B}$ be the matrix with rows from 1 to the $\min\{c-1, c'-1\}$ and columns one to $\min\{\text{end}_B, \text{end}_{B'}\}$. The solution of the computation is a pair of strings $(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$, the prefixes of the two computed strings until $\text{end}_{\tilde{B}}$. The value of (ζ, ζ') is $\text{cost}_{\tilde{B}}(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$.

In the update phase, we compute the value and the pair of strings of the DP cell (ζ, ζ') as follows. We inductively assume that all DP cells for predecessors of (ζ, ζ') have been updated already. We try all predecessor pairs of DP cells and keep the one that gives the best result. Let $(\overline{\zeta}, \overline{\zeta}')$ be a predecessor of (ζ, ζ'). By symmetry, we assume without loss of generality that $b' < b$. There are two cases how the two pairs interact. The first case is $\zeta = \overline{\zeta}$. We run SWC$_{\varepsilon^3}$ on the columns $\text{end}_{\overline{B}'} + 1$ to end_B with the parameters from (ζ, ζ') (see initialization). To obtain the full solution, we append the computed string for B' to the string $\sigma'_{\zeta,\overline{\zeta}'}$ (which is one of the solution strings of the predecessor pair). Let $\tilde{B}$ be the matrix with rows from 1 to the $\min\{c-1, c'-1\}$ and columns one to $\text{end}_{B'}$. The solution of the computation is a pair of strings $(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$, the prefixes of the two computed strings from column one to $\text{end}_{\tilde{B}}$. The potential new value of (ζ, ζ') is $\text{cost}_{\tilde{B}}(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$. We replace the stored solution with the potential new solution if the cost has decreased.

The second case is $\zeta' = \overline{\zeta}$. This case is the crux of the joint DP, since we have a "switch" of the role of σ and σ'. We run SWC$_{\varepsilon^3}$ on the columns $\text{end}_{\overline{B}}$ to $\text{end}_{B'}$ with the parameters from (ζ, ζ') (see initialization). To obtain the full solution, we then append the computed string for B to the string $\sigma_{\overline{\zeta},\zeta'}$ (which is one of the solution strings of the predecessor pair). Let $\tilde{B}$ be the matrix with rows from 1 to the $\min\{c-1, c'-1\}$ and columns one to $\text{end}_{B'}$. The solution of the computation is a pair of strings $(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$, the prefixes of the two computed strings until $\text{end}_{\tilde{B}'}$. The potential new value of (ζ, ζ') is $\text{cost}_{\tilde{B}}(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$. We replace the stored solution with the potential new solution if the cost has decreased.

For the last strings, we additionally consider special cells that are defined as before, but with $c = n$ or $c' = n$. Intuitively, we use these cells when only at most $1/\varepsilon^4$ rows of $\tau(M)$ or $\tau'(M)$ are left. For pairs of cells containing such ζ or ζ', our computation considers the optimal solution within the computation instead of $\text{SWC}_{\varepsilon^3}$.

▶ **Theorem 16.** *The algorithm* $\text{SWC}^{\sigma,\sigma'}$ *is a PTAS for SWC-instances.*

3 Subinterval-free instances

We show how to generalize the results of the previous section in order to handle instances where no interval of a string s is a proper subinterval of a string s' and thus show Theorem 2. To this end, we first show how to handle the rooted version of sub-interval free instances, where there is one column j such that each string of the instance crosses j.

We order the rows of a subinterval-free instance M from top to bottom such that for each pair i, i' of rows with the binary part of i starting on the left of the binary part of i', i is above i'. In other words, the binary strings are ordered from top to bottom with increasing starting position (i.e., column). Observe that the sub-string freeness property ensures that the last binary entry of i' is not on the left of the last binary entry of i.

▶ **Lemma 17.** *Let M be a* GAPLESS-MEC *instance such that no string is the substring of another string. Furthermore we assume that there is a column j of M such that each string of the instance crosses j. Then there is a PTAS for M.*

General sub-interval-free instances. We use Lemma 17 to handle general sub-interval free instances. Instead of a single column j crossed by all strings, we determine a sequence $q = (q_1, q_2, \dots)$ of columns with the property that each string crosses exactly one of them. Let s_1 be the first string in M. Then we choose q_1 to be the column of the last entry of s_1.

We recursively specify the remaining columns. For a given j such that we know q_j, let s_i be the last (i.e., bottom-most) string that crosses q_j. Then we choose q_{j+1} to be the last (i.e., rightmost) column of string s_{i+1}.

A simple induction shows that by the no-substring property and the chosen order of strings, each string crosses at least one column of q and none of them crosses more than two. In particular, for each j, the solution on the left hand side of q_j depends on rows of M disjoint from the rows that determine the solution on the right hand side of q_{j+1}.

In order to combine the solution on the right hand side of q_j with the solution on the left hand side of q_{j+1}, we introduce a notion of dominance.

▶ **Definition 18** (Dominance)**.** We say that a submatrix V_1 of M τ-dominates a submatrix V_2 of M if for each column c that is in both V_1 and V_2, either at least one of the two matrices has no binary entries or the number of binary entries in $\tau(V_1)$ is at least $1/\varepsilon^2$ times the number in $\tau(V_2)$. We say that V_1 is τ-dominant over V_2 for a column c, if the one column submatrix of V_1 determined by c dominates V_2. We analogously define τ'-dominance.

Consider a submatrix $\overrightarrow{V}$ of M that only contains rows that cross q_i and a submatrix $\overleftarrow{V}$ of M that only contains rows that cross q_{i+1}. We observe that if $\overrightarrow{V}$ is τ-dominant over $\overleftarrow{V}$

for some column c, it is also τ-dominant for all columns on the left hand side of c: until q_i is reached, when moving to the left the number of binary entries of $\tau(\overrightarrow{V})$ increases and the number of binary entries of $\tau(\overleftarrow{V})$ decreases. Analogously, if $\overleftarrow{V}$ is τ-dominant over $\overrightarrow{V}$ for some column c, it is also τ-dominant for all columns on the right hand side of c.

We therefore have a possibly empty interval I without τ-dominance such that the columns of $\overrightarrow{V}$ on the left hand side of I are τ-dominant and the columns of $\overleftarrow{V}$ on the right hand side of I are τ-dominant. (See also Figure 1.)

▶ **Definition 19** (Dominance region). The *dominance region* of $\overrightarrow{V}$ with respect to $\overleftarrow{V}$ is the set of columns where $\overrightarrow{V}$ is dominant over $\overleftarrow{V}$, and vice versa.

Within the dominance region, our old DP can simply compute solutions without considering interferences: the dominated set of rows is small enough to be ignored, applying Lemma 9.

Within the interval I, the DP cells on both sides of I have to "cooperate." We obtain a Binary-MEC block in the middle with additional rows on the top and bottom. This sub-instance can be solved directly.

4 A QPTAS for general instances

To solve the general instances, the main observation is that we divide the rows into their at most $\log_2(m)$ length classes Λ_i, and the ith length class Λ_i is the set of all strings of length ℓ with $\ell \in (m/2^{i+1}, m/2^i]$. First we present an algorithm to solve each length class Λ_i separately by constructing their corresponding columns.

4.1 Length classes

We show how we can handle length classes of strings. To this end, let us assume w.l.o.g. that m (i.e., the number of columns in M) is a power of 2. Then for each $i \geq 0$, the ith length class Λ_i is the set of all strings of length ℓ with $\ell \in (m/2^{i+1}, m/2^i]$. We observe the following known property of length classes.

▶ **Lemma 20.** *For each $i \geq 0$ there is a set $q_i = \{q_{i,1}, q_{i,2}, \dots\}$ of columns such that (a) each string in Λ_i crosses at least one column from q_i and (b) no string from Λ_i crosses more than two columns from q_i. Furthermore, we can choose the sets such that $q_i \subseteq q_{i+1}$.*

Proof. At level i, for each k with $1 \leq k \leq 2^{i+1}$ we select the column with index $k \cdot m/2^{i+1}$. We observe that the distance between two consecutive columns from q_i is $m/2^{i+1}$, which matches the shortest length of strings in Λ_i: if a minimal string starts right after a column of q_i, its last entry will cross the next column of q_i.

Since strings do not start before column 1 and column m is contained in each q_i, claim (a) follows. To see (b), observe that a maximum length string of Λ_i is at most $m/2^i$. Let j be an index. The number of columns from $q_{i,j}$ to the column right before $q_{i,j+1}$ and from $q_{i,j+1}$ to right before q_{i+2} are exactly $m/2^{i+1}$. If the string starts directly at a column $q_{i,j}$ from q_i, it would cross column $q_{i,j+1}$ and end right before column $q_{i,j+2}$. The last claimed property follows directly from the construction of the sets q_i. ◀

For each i, we now separate Λ_i into two sub-instances. One sub-instance Λ'_i is formed by those rows from Λ_i that only cross one column of q_i and the second sub-instance Λ''_i is formed by those rows that cross exactly two columns of Λ_i.

▶ **Definition 21** (DP for a length class Λ_i)**.** For each index j let ξ'_j be the sets of DP cells for Λ'_i and for the odd indices j let ξ''_j be the set of cells for Λ''_i. We define a super-cell that starts in j, $(Z'_j, Z''_j, Z'_{j+1}, Z''_{j+2}) \in \xi'_j \times \xi''_j \times \xi'_{j+1} \times \xi''_{j+2}$ and the super-cell that ends in j, $(Z'_{j-1}, Z''_{j-2}, Z'_j, Z''_j) \in \xi'_{j-1} \times \xi''_{j-2} \times \xi'_j \times \xi''_j$.

▶ **Lemma 22.** *There is a QPTAS for* Gapless-MEC *if all strings are in the same class* Λ_i.

4.2 The general QPTAS

Finally we combine our insights to an algorithm for general instances by combining different length classes. For different length classes Λ_i, we construct their corresponding columns as explained in the previous section. The main idea is that for each column j, we only have to consider those quadruple of super-cells according to Definition 21 that cross j from all the length classes simultaneously. We therefore consider at most $O(\log(n))$ quadruples of super-cells simultaneously. In the dynamic program, we consider a joint quadruple of super-cells from all the length classes. Then the overall complexity of a joint cell is quasi-polynomial: the number of different cells is $\left(n^{O(\log n)}\right)^{O(\log n)} = n^{O(\log^2 n)}$.

Let $Q_{i,j}$ be the set of quadruples of length class i crossing column j such that the strings are ordered from shortest length class to the longest. For each length class i, a quadruple $q \in Q_{i,j}$ is the set of rows starting at j, cross j, or end in j. If j is the index of $q_{i,\ell}$, the quadruple q starts in j if it is formed by cells $(Z'_\ell, Z''_\ell, Z'_{\ell+1}, Z''_{\ell+2})$ and ends in j if it is formed by $(Z'_{\ell-1}, Z''_{\ell-2}, Z'_\ell, Z''_\ell)$ (see Definition 21). If j lies between $q_{i,\ell}$ and $q_{i,\ell+1}$, j crosses those quadruples that contain Z'_ℓ and $Z'_{\ell+1}$. If none of the cases are true, we do not consider q in the cells for column j.

Let us consider a $\log(n)$ vector of quadruples v, with one quadruple $Q_{i,j}$ for each i and, consider quadruples starting at, ending at, or crossing column j for length class i. We require that if for some i, the quadruple $q \in Q_{i,j}$ ends at j, then for all the length classes Λ_k with $k > i$ the same condition holds (with index larger than i). This also implies that if for some i, the quadruple of length class i starts at j, then the same also holds for all quadruples of shorter length classes (with index larger than i). In particular, in order to be able to combine neighboring vectors of quadruples, we do not allow to mix starting and ending quadruples. Let ϕ be the set of all $\log(n)$ vectors of tuples as described above (with one tuple of each length class). The tuple for each length class is defined as in Lemma 22 and the DP for general instances follows the ideas of Lemma 22: We move from left to right column by column. In the initialization step, the joint DP cell is initialized based on Algorithm 1 using ϕ. We guess the blocks, chunks and selections from each length class and consider them jointly in a DP cell.

For column j, let us consider a vector $v \in \phi$. We distinguish whether v has starting or ending quadruples. (One of the two cases must apply due to the shortest length class.) For a $v \in \phi$ with starting quadruples, let d be the smallest number such that there is a quadruple of length class d starting at j. To compute v we consider all $v' \in \phi$ with the following properties. (a) v' has the same quadruples for all length classes $d' < d$ and (b) for $d' \geq d$, the right hand sides of the quadruples of length class d' in v' compatible the left hand sides of the quadruples of v. The super-cells from the left and right hand side are compatible if the intersecting strings from the left and right hand side are assigned to the same types of solution string σ or σ'.

For a $v \in \phi$ with ending quadruples, let d be the smallest number such that there is a quadruple of length class d ending at $j - 1$. (In the very first column of the instance, we do not need this value.) To compute v we consider all $v' \in \phi$ with the following properties. (a)

v' has the same quadruples for all length classes $d' < d$ and (b) for $d' \geq d$, the right hand sides of the quadruples of length class d' in v' match the left hand sides of the quadruples of v in column $j-1$. Then the value of v is the sum of the minimum value over all such v' and the number of errors in column j obtained by applying $\mathrm{SWC}_{\varepsilon^3}$ exactly as in the proof of Lemma 22.

The approximation ratio follows by arguing that the expected number of errors at each column is at most $(1+O(\varepsilon))$ of OPT (see Lemma 22). This finishes the proof of Theorem 1.

References

1 Noga Alon and Benny Sudakov. On two segmentation problems. *Journal of Algorithms*, 33(1):173–184, 1999.

2 Paola Bonizzoni, Riccardo Dondi, Gunnar W Klau, Yuri Pirola, Nadia Pisanti, and Simone Zaccaria. On the minimum error correction problem for haplotype assembly in diploid and polyploid genomes. *Journal of Computational Biology*, 2016.

3 Rudi Cilibrasi, Leo van Iersel, Steven Kelk, and John Tromp. The Complexity of the Single Individual SNP Haplotyping Problem. *Algorithmica*, 49(1):13–36, aug 2007. `doi:10.1007/s00453-007-0029-z`.

4 Uriel Feige. NP-hardness of hypercube 2-segmentation. *CoRR*, abs/1411.0821, 2014.

5 Pierre Fouilhoux and A. Ridha Mahjoub. Solving VLSI design and DNA sequencing problems using bipartization of graphs. *Computational Optimization and Applications*, 51(2):749–781, 2012. `doi:10.1007/s10589-010-9355-1`.

6 Shilpa Garg, Marcel Martin, and Tobias Marschall. Read-based phasing of related individuals. *Bioinformatics*, 32(12):i234–i242, 2016.

7 Dan He, Arthur Choi, Knot Pipatsrisawat, Adnan Darwiche, and Eleazar Eskin. Optimal algorithms for haplotype assembly from whole-genome sequence data. *Bioinformatics*, 26(12):i183–i190, 2010. `doi:10.1093/bioinformatics/btq215`.

8 Yishan Jiao, Jingyi Xu, and Ming Li. On the k-closest substring and k-consensus pattern problems. In *CPM*, volume 3109 of *Lecture Notes in Computer Science*, pages 130–144. Springer, 2004.

9 Jon M. Kleinberg, Christos H. Papadimitriou, and Prabhakar Raghavan. Segmentation problems. In *STOC*, pages 473–482. ACM, 1998.

10 Jon M. Kleinberg, Christos H. Papadimitriou, and Prabhakar Raghavan. Segmentation problems. *J. ACM*, 51(2):263–280, 2004.

11 Danny Leung, Inkyung Jung, Nisha Rajagopal, Anthony Schmitt, Siddarth Selvaraj, Ah Young Lee, Chia-An Yen, Shin Lin, Yiing Lin, Yunjiang Qiu, et al. Integrative analysis of haplotype-resolved epigenomes across human tissues. *Nature*, 518(7539):350–354, 2015.

12 Ming Li, Bin Ma, and Lusheng Wang. Finding similar regions in many sequences. *J. Comput. Syst. Sci.*, 65(1):73–96, 2002.

13 Rafail Ostrovsky and Yuval Rabani. Polynomial-time approximation schemes for geometric min-sum median clustering. *J. ACM*, 49(2):139–156, 2002.

14 Murray Patterson, Tobias Marschall, Nadia Pisanti, Leo van Iersel, Leen Stougie, Gunnar W. Klau, and Alexander Schönhuth. WhatsHap: Weighted haplotype assembly for future-generation sequencing reads. *Journal of Computational Biology*, 22(6):498–509, feb 2015. `doi:10.1089/cmb.2014.0157`.

15 Yuri Pirola, Simone Zaccaria, Riccardo Dondi, Gunnar W. Klau, Nadia Pisanti, and Paola Bonizzoni. HapCol: accurate and memory-efficient haplotype assembly from long reads. *Bioinformatics*, page btv495, aug 2015. `doi:10.1093/bioinformatics/btv495`.

16 Jan Remy and Angelika Steger. Approximation schemes for node-weighted geometric steiner tree problems. *Algorithmica*, 55(1):240–267, 2009.

17 Matthew W Snyder, Andrew Adey, Jacob O Kitzman, and Jay Shendure. Haplotype-resolved genome sequencing: experimental methods and applications. *Nature Reviews Genetics*, 16(6):344–358, 2015.

18 Ryan Tewhey, Vikas Bansal, Ali Torkamani, Eric J Topol, and Nicholas J Schork. The importance of phase information for human genomics. *Nature Reviews Genetics*, 12(3):215–223, 2011.

19 Sharon Wulff, Ruth Urner, and Shai Ben-David. Monochromatic bi-clustering. In *ICML (2)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 145–153. JMLR.org, 2013.

FPT Algorithms for Embedding into Low Complexity Graphic Metrics

Arijit Ghosh
The Institute of Mathematical Sciences, HBNI, Chennai, India
arijitiitkgpster@gmail.com

Sudeshna Kolay
Eindhoven University of Technology, Eindhoven, Netherlands
s.kolay@tue.nl

Gopinath Mishra
Indian Statistical Institute, Kolkata, India
gopianjan117@gmail.com

Abstract

The Metric Embedding problem takes as input two metric spaces (X, D_X) and (Y, D_Y), and a positive integer d. The objective is to determine whether there is an embedding $F : X \to Y$ such that the distortion $d_F \leq d$. Such an embedding is called a *distortion d embedding*. In parameterized complexity, the Metric Embedding problem is known to be W-hard and therefore, not expected to have an FPT algorithm. In this paper, we consider the Gen-Graph Metric Embedding problem, where the two metric spaces are graph metrics. We explore the extent of tractability of the problem in the parameterized complexity setting. We determine whether an unweighted graph metric (G, D_G) can be embedded, or bijectively embedded, into another unweighted graph metric (H, D_H), where the graph H has low structural complexity. For example, H is a cycle, or H has bounded treewidth or bounded connected treewidth. The parameters for the algorithms are chosen from the upper bound d on distortion, bound Δ on the maximum degree of H, treewidth α of H, and the connected treewidth α_c of H.

Our general approach to these problems can be summarized as trying to understand the behavior of the shortest paths in G under a low distortion embedding into H, and the structural relation the mapping of these paths has to shortest paths in H.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Metric spaces, metric embedding, FPT, dynamic programming

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.35

Related Version A full version of the paper is available at [8], `https://arxiv.org/abs/1801.03253`.

1 Introduction

Given metric spaces (X, D_X) and (Y, D_Y), an *embedding* $F : X \to Y$ is an injective mapping from X to Y. The *expansion* e_F and *contraction* c_F of F are defined as $e_F = \max_{x_1, x_2 (\neq x_1) \in X} \frac{D_Y(F(x_1), F(x_2))}{D_X(x_1, x_2)}$ and $c_F = \max_{x_1, x_2 (\neq x_1) \in X} \frac{D_X(x_1, x_2)}{D_Y(F(x_1), F(x_2))}$, respectively. The *distortion* $d_F = e_F \cdot c_F$. Observe that $d_F \geq 1$. An embedding $F : X \to Y$ is *non-contracting* if $c_F \leq 1$.

The problem of low distortion embedding of a metric space into a simple metric space has been extensively studied in Mathematics and Computer Science (see [1, 11, 13, 14, 15]). Low distortion embedding algorithms have also found wide applications in other problems like

© Arijit Ghosh, Sudeshna Kolay, and Gopinath Mishra;
licensed under Creative Commons License CC-BY

Sparsest Cut, Nearest Neighbor Search, Clustering, Multicommodity Flow, Multicut, Small Balanced Separators (see [1, 9, 10, 13, 15]).

The need to obtain small distortion embeddings into simpler spaces naturally led to the question of finding a minimum distortion embedding of (X, D_X) into (Y, D_Y) when both the metric spaces have shortest path metrics on graphs with positive weights, and (Y, D_Y) has a simple topology as in paths, cycles, trees etc. Kenyon et al. [12] showed that this problem is APX-hard even when both the graphs are unweighted, have the same number of vertices, and one of the graphs is a simple wheel graph. Badoiu et al. [3] also proved APX-hardness when both the graphs are unweighted and (Y, D_Y) is the metric space of a path. Badoiu et al. [2] showed that computing the minimum distortion is hard to approximate up to a factor polynomial in $|X|$, even when (X, D_X) is a weighted tree with polynomial spread and (Y, D_Y) is a path. Fellows et al. [7] showed that the problem of embedding a weighted graph metric into a path with distortion at most $d > 2$ is NP-complete.

Badoiu et al. [3] gave the first algorithm for deciding if an unweighted graph metric has a non-contracting embedding into a path with distortion d. The running time of their algorithm was $n^{4d+2} \cdot d^{O(1)}$, where n denotes the number of vertices in the graph. Fellows et al. [7] gave the first fixed parameter tractable(FPT) algorithm with running time $O\left(n\, d^4(2d+1)^{2d}\right)$ for finding a non-contracting embedding of an n vertex unweighted graph metric into a path with distortion at most d (d is the parameter of the algorithm). They also showed that their FPT algorithm can be extended to get an FPT algorithm for the case of non-contracting embeddings of weighted graphs into paths, where the parameters to the algorithm are both the distortion and the maximum weight of an edge in the graph. Nayyeri et al. [16] gave improved exact algorithms for embedding weighted path metrics into weighted paths.

Kenyon et al. [12] gave the first FPT algorithm for finding a bijective embedding f of an unweighted graph metric on n vertices into a tree with maximum degree bounded by Δ in $O(n^2 \cdot 2^{\Delta^{\mu^3}})$ time, where $\mu = \max\{e_f, c_f\}$. Fellows et al. [7] extended this result to give an algorithm for the problem of finding a non-contracting embedding of unweighted graphs into bounded degree trees with distortion at most d in $O(n^2 \cdot |V(T)|) \cdot 2^{O((5d)^{\Delta^{d+1}} \cdot d)}$ time, where $V(T)$ denotes the vertex set of the tree and where the maximum degree in T is bounded by Δ. In a follow-up paper, Nayyeri et al. [17] gave the first $(1+\epsilon)$-approximation algorithm to embed weighted graphs with spread Σ into graphs on m vertices with bounded treewidth α and doubling dimension λ in $m^{O(1)} \cdot n^{O(\alpha)} \cdot (d_{opt}\Sigma)^{\alpha \cdot (1/\epsilon)^{\lambda+2} \cdot \lambda \cdot (O(d_{opt}))^{2\lambda}}$ time, where d_{opt} denotes the minimum distortion.

Our Contributions. In this paper, we further investigate the problem of embedding a general graph metric (G, D_G) into a low complexity graph metric (H, D_H) with distortion at most d. We will denote by n and N the number of vertices in graphs G and H, respectively. Also, we denote distortion by d, and the maximum degree of H by Δ. We denote by ℓ_g the length of a largest induced cycle, or geodesic cycle, in H. We approach the metric embedding problem by trying to understand the behavior of the shortest paths in G under a low distortion embedding into H, and what relation the mapping of these paths has to shortest paths in H. Careful analysis of this connection helps us solve a number of problems in this area, in the parameterized setting. **All the algorithmic results mentioned below are regarding non-contracting bounded distortion embeddings. However, all these results can be extended to find bounded distortion embeddings, without the assumption on non-contraction. For all the results, if the running time of the stated algorithm is T, then the running time of finding a bounded distortion embedding will be $(nN)^{O(1)} \cdot T$.**

We first begin by proving the following open question in [7]. Independently, a similar result on the same problem was obtained in same time by Carpenter et al. [4].

▶ **Theorem 1.1.** *Given an undirected unweighted graph G on n vertices, a cycle C and a distortion parameter d, there exists an algorithm that either finds a non-contracting distortion d embedding of G into C or decides that there does not exist such an embedding in $O(n^3 \cdot d^{2d+3} \cdot (4d(2d+2))^{4d+4})$ time.*

Due to the existence of the large geodesic cycle that is the graph H, techniques from the previous papers, like *pushing embeddings* [7], do not work and some new ideas are required to solve this problem. Moreover, our FPT algorithm can be extended to the case when the input graph G is a weighted graph, and we can parameterize by the distortion d and the maximum edge weight in G. We also show that the problem is NP-Complete when we do not take the maximum edge weight as a parameter for any distortion $d > 2$.

Observe that the *treewidth* of a cycle is 2, but the *connected treewidth* of a cycle is $\Omega(n)$ (see the definitions of treewidth and connected treewidth in Section 2). These two parameters (treewidth and connected treewidth of graphs) play important roles in this paper. In this direction, we first extend the result of Kenyon et al. [12] for bijection into bounded degree trees.

▶ **Theorem 1.2.** *Let G, H be two given graphs such that $|V(G)| = |V(H)| = n$, the maximum degree of H is Δ and the graph H has treewidth $tw(H) \leq \alpha$. Then there exists an algorithm that either finds a bijective non-contracting distortion d embedding of G into H or decides no such embedding exists in $O(\alpha^2 n^{\alpha+3}) \cdot \Delta^{d+1} \cdot (\alpha\Delta^{d+1})^{\Delta^{O(\alpha d^2)}}$ time.*

Note that the algorithm in Theorem 1.2 is not an FPT algorithm if $tw(H)$ is an input parameter to the problem. Therefore, it is natural to ask if we can still get FPT algorithms for a more general case, where $tw(H)$ is considered as a parameter instead of a constant. In this context, we prove the following result:

▶ **Theorem 1.3.** *Let G, H be two given graphs with n and N vertices, respectively, such that the maximum degree of H is Δ, treewidth $tw(H) \leq \alpha$ and the length of the longest geodesic cycle in H is ℓ_g. Then there exists an algorithm that either finds a non-contracting distortion d embedding of G into H or decides no such embedding exists in running time $O(n^2 \cdot N) \cdot (\alpha \cdot \Delta^{d+1})^{\Delta^{O(\mu \cdot d + d^2)}} \cdot 2^{O((4(\mu+d))^{\alpha^2 \cdot \Delta^{d+1}})}$, where $\mu = 4(\alpha + \binom{\alpha}{2}(\ell_g(\alpha - 2) - 1))$.*

This result crucially uses the result in [6] that a graph has bounded connected treewidth if and only if the graph has bounded treewidth and no long geodesic cycle. It is to be noted that a wheel graph has constant connected treewidth, and by a result in [12], embedding into wheel graphs is NP-hard even when the distortion $d = 2$. However, when the wheel graph has bounded degree, then the number of vertices in the wheel graph becomes bounded, and we obtain a trivial FPT algorithm parameterized by the degree and the distortion d. This motivated us to consider the above variant of metric embedding. Our FPT algorithm extends the result of Fellows et al. [7] for embedding into trees with bounded degree. Controlling the behavior of shortest paths in the graph G under a low distortion embedding into the class of graphs with bounded degree and bounded connected treewidth is algorithmically considerably harder than the case of bounded degree trees.

We also investigate bounded distortion embedding into *generalized theta graphs*: defined by the union of k internally vertex-disjoint paths all of which have common endpoints s and t. We prove the following result for generalized theta graphs.

▶ **Theorem 1.4.** Metric Embedding *into generalized theta graphs is FPT parameterized by distortion d and number k of $s-t$ paths. The algorithm runs in time $O(N) + n^5 \cdot k^{2k+1} \cdot (kd+1)^{(2d)^{O(kd)}} \cdot d^{O(d^2)}$, where n and N are the number of vertices in the input and output graph metrics, respectively.*

As mentioned earlier, it was shown in [6] that a graph has bounded connected treewidth if and only if the graph has bounded treewidth and no long geodesic cycle. In general, embedding into graphs with large geodesic cycles is not amenable to known algorithmic techniques in the parameterized settings. Intuitively, all known techniques for designing FPT algorithms in this area used the fact that if a low distortion embedding F exists, then the embedding of a shortest path between two vertices $u, v \in V(G)$ and the shortest path in H between $F(u)$ and $F(v)$ are somewhat structurally related. With the presence of large geodesic cycles this structural relation may completely break down: although the two paths have similar lengths, structurally they could be completely different. This poses a problem for designing dynamic programming algorithms, a staple for FPT algorithms in this area. The class of generalized theta graphs has treewidth 2, but may have large geodesic cycles. Hence, these graphs are more general than cycles and have constant treewidth, but they do not have bounded connected treewidth. Even for this very structured graph class, by virtue of the graphs having long geodesic cycles, we needed to develop completely new ideas in order to find low distortion embeddings into generalized theta graphs via FPT algorithms. The problem arises from the fact that any two geodesic cycles of a generalized theta graph intersect at at least two vertices, and there are many pairs of geodesic cycles with large intersections. Our algorithm is still a dynamic programming algorithm, but a more involved one. The way to work around the apparent barriers is to investigate more closely the structural properties of an input graph G that can be embedded with small distortion into a generalized theta graph. Independently, a generalization of this result was obtained in same time by Carpenter et al. [4]. We would like to mention that our algorithms for embedding into cycles and generalized theta graphs have better time complexity.

This is an extended abstract. For full details please refer to the full version of the paper [8].

2 Preliminaries

General Notation. We denote $\{1, \ldots, t\}$ as $[t]$. For a set S, $|S|$ denotes the number of elements present in S. Given a function $f : U' \to D'$ and a function $F : U \to D$, where $U' \subseteq U$ and $D' \subseteq D$, we say that F *extends* f if for all $x \in U'$, $F(x) = f(x)$. For a set of functions $\Pi = \{f_i : A_i \to B_i, i \in [t]\}$ such that for any $i, j \in [t]$, $x \in A_i \cap A_j$ implies $f_i(x) = f_j(x)$, we define $\Phi_\Pi : \bigcup_{i=1}^{t} A_i \to \bigcup_{i=1}^{t} B_i$ such that $\Phi_\Pi(x) = f_i(x)$ for $i \in [t], x \in A_i$.

A graph is denoted by G while its vertex set and edge set are denoted by $V(G)$ and $E(G)$, respectively. We denote the set of neighbours of a vertex $v \in V(G)$ as $N_G(v)$. The degree of a vertex $v \in V(G)$ is denoted as $\mathsf{deg}_G(v)$. We also define $\Delta(G) = \max_{v \in V(G)} \mathsf{deg}_G(v)$. We also define the set $B(v, r) = \{u \in V(H) \mid D_H(u, v) \leq r\}$, and refer to it as an r-ball around v. For a subgraph G' of G, $v \in V(G) \setminus V(G')$ is said to be a neighbour of G' if there is a vertex $u \in V(G')$ such that $(u, v) \in E(G)$. A subgraph G' of G is said to be an *induced subgraph* if $E(G') = \{(u, v) \in E(G) | u, v \in V(G')\}$. An induced cycle in a graph is also called a *geodesic cycle*.

A *generalized theta graph* is the union of k paths $\mathcal{P} = \{P_1, P_2, \dots P_k\}$ such that the endpoints of all the paths are two vertices s and t, while every pair of paths are internally vertex and edge disjoint. Such a graph will also be referred to as a generalized theta graph defined at s,t, and the family $\mathcal{P}$ is said to define the generalized theta graph.

Treewidth. A *tree decomposition* [5] of a graph G is a tuple $\mathcal{T} = (T, \{X_{\mathbf{u}}\}_{\mathbf{u} \in V(T)})$, where T is a tree in which each vertex $\mathbf{u} \in V(T)$ has an assigned set of vertices $X_{\mathbf{u}} \subseteq V(G)$ (called a bag) such that the following properties hold: (i) $\bigcup_{\mathbf{u} \in V(T)} X_{\mathbf{u}} = V(G)$, (ii) for any $(x, y) \in E(G)$, there exists a $\mathbf{u} \in V(T)$ such that $x, y \in X_{\mathbf{u}}$, (iii) if $x \in X_{\mathbf{u}}$ and $x \in X_{\mathbf{v}}$, then $x \in X_{\mathbf{w}}$ for all $\mathbf{w}$ on the path from $\mathbf{u}$ to $\mathbf{v}$ in T.

The *treewidth* $tw_{\mathcal{T}}$ of a tree decomposition $\mathcal{T}$ is the size of the largest bag of $\mathcal{T}$ minus one. A graph may have several distinct tree decompositions. The treewidth $tw(G)$ of a graph G is defined as the minimum of treewidths over all possible tree decompositions of G. Note that for the tree T of a tree decomposition, we denote a vertex of $V(T)$ in bold font.

A tree decomposition $\mathcal{T} = (T, \{X_{\mathbf{u}}\}_{\mathbf{u} \in V(T)}))$ is called a *nice tree decomposition* if T is a tree rooted at some node $\mathbf{r}$ where $X_{\mathbf{r}} = \emptyset$, each node of T has at most two children, and each node is of one of the following kinds: (i) **Introduce node**: a node $\mathbf{u}$ that has only one child $\mathbf{u}'$ where $X_{\mathbf{u}} \supset X_{\mathbf{u}'}$ and $|X_{\mathbf{u}}| = |X_{\mathbf{u}'}| + 1$, (ii) **Forget vertex node**: a node $\mathbf{u}$ that has only one child $\mathbf{u}'$ where $X_{\mathbf{u}} \subset X_{\mathbf{u}'}$ and $|X_{\mathbf{u}}| = |X_{\mathbf{u}'}| - 1$, (iii) **Join node**: a node $\mathbf{u}$ with two children $\mathbf{u}_1$ and $\mathbf{u}_2$ such that $X_{\mathbf{u}} = X_{\mathbf{u}_1} = X_{\mathbf{u}_2}$, (iv) **Leaf node**: a node $\mathbf{u}$ that is a leaf of T, and $X_{\mathbf{u}} = \emptyset$.

One can show that a tree decomposition of width w can be transformed into a nice tree decomposition of the same width w and with $O(w|V(G)|)$ nodes, see e.g. [5]. For a node $\mathbf{u} \in V(T)$, let $T_{\mathbf{u}}$ denote the subtree rooted at $\mathbf{u}$ and $H_{\mathbf{u}}$ denote the subgraph induced by $\bigcup\limits_{\mathbf{v} \in V(T_{\mathbf{u}})} X_{\mathbf{v}}$. The set $\mathcal{B}(\mathbf{u}, r) = \bigcup\limits_{x \in X_{\mathbf{u}}} B(x, r)$

A *connected tree decomposition* is a tree decomposition where the vertices in every bag induce a connected subgraph of G [6]. The *connected treewidth* $ctw(G)$ of a graph G is defined as the minimum of treewidths over all possible connected tree decompositions of G.

Given a graph G, the function $D_G : V(G) \times V(G) \to \mathbb{R}$ is the shortest distance function defined on G; for any pair $u, v \in V(G)$, $D_G(u, v)$ is the length of the shortest path between u and v in the graph G. When we talk of a graph metric, then we denote it as the tuple (G, D_G). In this paper, unless otherwise mentioned, a graph metric is that of an *unweighted undirected graph.*

Metric Embedding. A metric embedding of a graph metric (G, D_G) into a graph metric (H, D_H) is a function $F : V(G) \to V(H)$. When the graph metrics are clear, we also use the terminology that the metric embedding is that of G into H, or that G is embedded into H. We also denote (G, D_G) as the *input metric space* and (H, D_H) as the *output metric space.* A non-contracting distortion d metric embedding implies that the expansion is at most d. Therefore, for any pair $u, v \in V(G)$, $D_G(u, v) \leq D_H(F(u), F(v)) \leq d \cdot D_G(u, v)$.

We consider the following two problems in this paper.

Gen-Graph Metric Embedding
Input: Two graph metrics (G, D_G) and (H, D_H), where G is a connected graph, and a positive integer d
Question: Is there a distortion d metric embedding of (G, D_G) into (H, D_H)?

Graph Metric Embedding
Input: Two graph metrics (G, D_G) and (H, D_H), where G is a connected graph, and a positive integer d
Question: Is there a non-contracting distortion d metric embedding of (G, D_G) into (H, D_H)?

The bijective versions of the above problems takes the same input but aims to determine whether the distortion d embedding is a bijective function. The Gen-Graph Metric Embedding problem or the Graph Metric Embedding problem for a graph class $\mathcal{G}$ is a variant where the output metric space (H, D_H) is such that $H \in \mathcal{G}$. In this extended abstract, we present results for the Graph Metric Embedding problem.

Parameterized Complexity. The instance of a *parameterized problem/language* is a pair containing the problem instance of size n and a positive integer k, which is called a parameter. The problem is said to be in FPT if there exists an algorithm that solves the problem in $f(k)n^{O(1)}$ time, where f is a computable function. Readers are requested to refer [5] for more details on Parameterized Complexity.

3 Graph Metric Embedding for Generalized Theta graphs

In this section, we design an FPT algorithm for embedding unweighted graphs into generalized theta graphs. Our FPT algorithm is parameterized by the distortion d and the number k of paths in the generalized theta graph. The strategy for the algorithm is still the same: that of putting together partial embeddings to obtain a non-contracting distortion d metric embedding. For this algorithm, we also observe structural properties of graphs that are embeddable into generalized theta graphs. We exploit these properties to obtain an FPT algorithm to compute a set of partial embeddings, and then use a dynamic programming algorithm to put together partial embeddings from the set to obtain the solution metric embedding. This makes the notion of partial embeddings more involved in this algorithm.

Let (G, D_G) be the graph metric that we want to embed into the graph metric (H, D_H). Here H is a generalized theta graph defined at s,t and let $\mathcal{P}$ be the family of $s-t$ paths that define H. To begin with, we try to guess the non-contracting distortion d embedding of (G, D_G) into (H, D_H), when restricted to a d-ball around s and around t.

▶ **Definition 3.1.** Let F be a non-contracting distortion d embedding of G into H. Define $B_s = \{v \in V(H) \mid D_H(v, s) \leq d\}$ and $B_t = \{v \in V(H) \mid D_H(v, t) \leq d\}$. For an embedding $F : V(G) \to V(H)$, define $\mathsf{Dom}_s^F = \{u \in V(G) \mid F(u) \in B_s\}$ and $\mathsf{Dom}_t^F = \{u \in V(G) \mid F(u) \in B_t\}$.

The following observation talks about the degree bound on the vertices of a graph that is embeddable into a generalized theta graph.

▶ **Observation 3.2.** If there exists a non-contracting distortion d embedding F of G into H, then:

(i) Each vertex in Dom_s^F can have degree at most $(k+1)d$. Similarly, each vertex in Dom_t^F can have degree at most $(k+1)d$,

(ii) All other vertices of G can have degree at most $2d$.

▶ **Observation 3.3.** The number of possible non-contracting distortion d embeddings of some $U \subseteq V(G)$ into $B_s \cup B_t$ is at most $n^2 \cdot (kd+1)^{(2d)^{O(kd)}}$.

We prove several properties of graphs that are embeddable into generalized theta graphs. For the given input graph G, let F be a non-contracting distortion d embedding and $\Psi : \mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F \to B_s \cup B_t$ be the restriction of F to $B_s \cup B_t$. Let $C_1, C_2, \ldots C_a$ be the components of $G \setminus (\mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F)$.

▶ Remark. For simplicity of the presentation, we will assume the following:
1. For all $i \in [a]$, we have $|C_i| > 2kd^2 + 1$, and
2. for all $j \in [k]$, we have $|P_j| > 2k(2kd^2 + 1) + 3d$.
The details of the general case is handled in the full version [8].

We will derive certain properties of G with the help of the embedding F. For each $i \in [k]$, let $P_i' = P_i \setminus (B_s \cup B_t)$. If P_i' is a non-empty path, let s_i be the endpoint of P_i' that has an edge to B_s while t_i be the endpoint of P_i' that has an edge to B_t. Let S_i (T_i) denote the set of vertices of Dom_s^F (Dom_t^F) that are mapped into P_i.

▶ **Observation 3.4.** Let F be a non-contracting distortion d embedding, $\Psi : \mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F \to B_s \cup B_t$ be the restriction of F to $B_s \cup B_t$, and $C_1, C_2, \ldots C_a$ be the components of $G \setminus (\mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F)$. Then each component of $G \setminus (\mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F)$ can have it's vertices mapped into exactly one P_i', $i \in [k]$. One the other hand, each P_i', $i \in [k]$, can have at most 2 connected components of $G \setminus (\mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F)$ mapped into it, in the non-contracting distortion d embedding F.

Thus, there can be at most $2k$ components of $G \setminus (\mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F)$.

▶ **Definition 3.5.** Let F be a non-contracting distortion d embedding. An empty subpath of F is a subpath of the generalized theta graph where none of the vertices have any preimage. If a path P_i', $i \in [k]$, has an empty subpath with one endpoint at t_i, then such a subpath is called a *t-empty subpath.* Similarly, if a path P_i' has an empty subpath with one endpoint at s_i, then such a subpath is called a *s-empty subpath.* If a path P_i' contains an empty subpath that coincides with neither s_i nor t_i, then such a subpath is called an *internal-empty subpath.* Finally, it is possible that the path P_i' itself is an empty subpath and then P_i' is called a *fully-empty subpath.*

Note that a path P_i' can have at most one empty subpath with respect to F. Similarly, we classify the components of $G \setminus (\mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F)$.

▶ **Definition 3.6.** Let F be a non-contracting distortion d embedding. A component in $G \setminus (\mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F)$ is called an *s-component* if it has neighbours to Dom_s^F and not to Dom_t^F. Similarly, we define a t-component. A *full component* is a component that has neighbours to both Dom_s^F and Dom_t^F.

Since F is a non-contracting distortion d embedding, the following observation is true.

▶ **Observation 3.7.** Let F be a non-contracting distortion d embedding. Any path P_i, $P_i' \neq \emptyset$, can be one of the following *forms*: **(i)** *form-1:* It has an s-component mapped into it by F, and a t-empty subpath, **(ii)** *form-2:* It has a t-component mapped into it by F, and an s-empty subpath, **(iii)** *form-3:* It has an s-component and a t-component mapped into it by F, and an internal-empty subpath, **(iv)** *form-4:* It has a full component mapped into it by F, and **(v)** *form-5:* It contains a fully-empty subpath.

If we refer P_i to be of *form-* ST , then P_i is of form-1 or form-2 or form-3. The objective is to find a non-contracting distortion d embedding F, if it exists. Although we do not know about F, we want to store a snapshot of F.

▶ **Definition 3.8.** A *configuration* $\mathcal{X}$ is a tuple $(\Psi, \mathcal{P}', \hat{\mathcal{P}})$ where:

(i) Let $U \subseteq V(G)$ be such that $G \setminus U$ creates a set of components $\{C_1, C_2, \ldots, C_a\}$, $a \leq 2k$. $\Psi : U \to B_s \cup B_t$ is a non-contracting distortion d embedding of U.

(ii) $\mathcal{P}' \subseteq \mathcal{P}$,

(iii) $\hat{\mathcal{P}}$ is a family of $|\mathcal{P} \setminus \mathcal{P}'|$ tuples such that for each path $P_i \in \mathcal{P} \setminus \mathcal{P}'$, there is a tuple $(\mathsf{form}_i, \mathcal{C}_{P_i}, \mathsf{comp}_i)$ with the following information: (a) form_i assigns the name of a form to P_i, (b) The set $\mathcal{C}_{P_i}$ is a set of at most 2 components of $G \setminus U$ that are assigned to P_i' and to no other $P_j', j \neq i$, (c) The function comp_i indicates for each $C \in \mathcal{C}_{P_i}$ whether it is an s-component or a t-component or full-component, with respect to Ψ.

(iv) $\bigcup_{P_i \in \mathcal{P} \setminus \mathcal{P}'} \mathcal{C}_{P_i}$ has all the components of $G \setminus U$.

For any fixed Ψ, the total number of configurations is $O(k^{2k})$. Next, we define feasible configurations that can be associated with metric embeddings.

▶ **Definition 3.9.** A configuration $\mathcal{X} = (\Psi, \mathcal{P}', \hat{\mathcal{P}})$ is said to be *feasible* with respect to a non-contracting distortion d embedding F of G into H if the following hold:

(i) $\Psi : \mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F \to B_s \cup B_t$ is the restriction of F to $\mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F$.

(ii) $P_i' = P_i \setminus (B_s \cup B_t)$ is empty for each $P_i \in \mathcal{P}' \subseteq \mathcal{P}$. ,

(iii) For each P_i that is non-empty with respect to F, $\hat{\mathcal{P}}$ contains a tuple $(\mathsf{form}_i, \mathcal{C}_{P_i}, \mathsf{comp}_i)$ with the following information: (a) form_i is the form of P_i in F, (b) The set $\mathcal{C}_{P_i}$ is the set of at most 2 components of $G \setminus U$ that are embedded into P_i' by F, (c) The function comp_i indicates for each $C \in \mathcal{C}_{P_i}$ whether it is an s-component or a t-component or full-component, with respect to F.

(iv) $\bigcup_{P_i \in \mathcal{P} \setminus \mathcal{P}'} \mathcal{C}_{P_i}$ has all the components of $G \setminus (\mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F)$.

We denote a configuration feasible with respect to F as $\mathcal{X}(F)$.

Next, we define the notion of a last vertex for a component of $G \setminus (\mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F)$ with respect to the embedding F.

▶ **Definition 3.10.** Let F be a non-contracting distortion d embedding. Let C be a j-component, $j \in \{s, t\}$. A vertex ℓ in C is the *last* vertex of C with respect to embedding F if $D_H(j, F(\ell)) \geq D_H(j, F(x))$ for all $x \in C$.

The following Lemma gives a bound on the potential last vertices of a component of $G \setminus (\mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F)$ if G is embeddable into H.

▶ **Lemma 3.11.** *Let $\mathcal{F}$ be a family of non-contracting distortion d embedding of G into H such that $\mathcal{X}(F_1) = \mathcal{X}(F_2)$ for any $F_1, F_2 \in \mathcal{F}$. Then for any form-* ST *path P_i and any $s(t)$-component $C \in \mathcal{C}_{P_i}$, there are $d^{O(d^2)}$ vertices that are candidates for being the last vertex of C with respect to some $F \in \mathcal{F}$.*

Next, we define the notion of a shortest embedding of a component in a path of $\mathcal{P}$.

▶ **Definition 3.12.** Let $\mathcal{Y}$ be a feasible configuration such that $\mathcal{Y} = \mathcal{X}(F)$ for a non-contracting distortion d embedding F. Let P_i be a form- ST path, $C \in \mathcal{C}_{P_i}$ be a s-component of $G \setminus (\mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F)$ and $\ell \in C$ be a candidate to be the last vertex of C with respect F.

Recall that S_i is the set of vertices of Dom_s^F that are mapped into P_i. Let $\mathcal{A}$ be a family of non-contracting and distortion d embedding of $C \cup S_i$ into P_i such that the following conditions hold: **(i)** $f_1|_{S_i} = f_2|_{S_i}$ for any $f_1, f_2 \in \mathcal{A}$, **(ii)** For each $f \in \mathcal{A}$, $f(x)$ is a vertex of P_i' for any $x \in C$, **(iii)** For each $f \in \mathcal{A}$, $F|_{C \cup S_i} = f$ and ℓ is the last vertex of C with respect to F, and **(iv)** For each $f \in \mathcal{A}$, for any $x \in \mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F$, the path between $f(\ell)$ and $f(x)$ is non-contracting with expansion at most d.

Then the *shortest* embedding of $C \cup S_i$ into P_i with respect to $\mathcal{Y}$ and ℓ, is an embedding $f \in \mathcal{A}$ such that $D_H(s, f(\ell)) \leq D_H(s, f'(\ell))$ for all $f' \in \mathcal{A}$. If C is a t-component, T_i is taken to be the set of vertices of Dom_t^F that are mapped into P_i and we can define the shortest embedding of $C \cup T_i$ with respect to $\mathcal{Y}$ and ℓ in a similar way.

We can extend the notion of shortest embedding of a component into a path of $\mathcal{P}$ to that of a non-contracting distortion d embedding of G into H that has shortest embeddings for all s-components and t-components.

▶ **Definition 3.13.** Let us consider a non-contracting distortion d embedding F of G into H. We say F is a *special embedding with respect to feasible configuration* $\mathcal{X}(F)$ if for every path P_i of form- ST and s (t)-component $C \in \mathcal{C}_P$, the following holds: $F|_{C \cup S_i}$ ($F|_{C \cup T_i}$) is the shortest embedding of $C \cup S_i$ ($C \cup T_i$) into P_i with respect to the feasible configuration $\mathcal{X}(F)$ and the last vertex of C with respect to F.

The next lemma shows that it is enough to look for a special embedding of G into H.

▶ **Lemma 3.14.** *If there exists a non-contracting distortion d embedding of G into H, then there exists a special embedding of G into H with respect to some configuration.*

Therefore, we have shown that if G is embeddable into H then it is enough to find a special embedding. We design an FPT algorithm for finding a special embedding.

Proof Sketch of Theorem 1.4. By Lemma 3.14, it is sufficient to look for special embedding with respect to some configuration. We find $\Delta(G)$ and if $\Delta(G) > (k+1)d$, then we report NO. This is correct by Observation 3.2.

We first compute $D_H(s, u)$ and $D_H(t, u)$ for all $u \in V(H)$. We store this distance information in a matrix $\mathcal{D}_{st}$, such that the look-up time for the distance from any $u \in V(H)$ to s or t is $O(1)$. Next, let us fix a non-contracting distortion d embedding Ψ of $U \subseteq V(G)$ into $B_s \cup B_t$ and a configuration $\mathcal{Y}$ containing Ψ. If the degree of any vertex in $G \setminus U$ is more than $2d$, then we decide that there does not exist any desired embedding with respect to $\mathcal{Y}$. Otherwise, we proceed as follows. Let F be the special embedding of G into H with respect to $\mathcal{Y}$ that we want to find, if one exists. Note that $U = \mathsf{Dom}_s^F \cup \mathsf{Dom}_t^F$.

(i) If a path P_i is of form-5, we don't have to do anything for that.

(ii) Let a path P_i be of form-4, and suppose $C \in \mathcal{C}_{P_i}$ is the only full component mapping into P_i. Then we find a non-contracting distortion d embedding f_C, if possible, of $C \cup S_i \cup T_i$ into P_i such that $f_C|_{S_i} = \Psi|_{S_i}$ and $f_C|_{T_i} = \Psi|_{T_i}$. Such an algorithm is described in the full version of the paper. If we cannot find such an embedding, then there does not exist any special embedding of G into H with respect to $\mathcal{Y}$.

(iii) Let P_i be a form- ST path and $C \in \mathcal{C}_{P_i}$ be an (a) s (t)-component. Without loss of generality, assume that C is an s-component. Here, our objective is to find the shortest embedding f of $C \cup S_i$ into P_i with respect to $\mathcal{Y}$ and some ℓ, where ℓ is the last vertex of C with respect to F. We guess a vertex $\ell \in C$, as the last vertex. By Lemma 3.11, the total number of candidates for the last vertex of C with respect to F is $d^{O(d^2)}$. It is easy to see that $|C \cup S_i| \leq D_H(f(\ell), f(a)) \leq 2d.|C \cup S_i|$. Thus, the length of the shortest embedding of $C \cup S_i$, where ℓ is the last vertex, is also in this range. For each possible length $|C \cup S_i| \leq \mathsf{len} \leq 2d.|C \cup S_i|$, we try to find a non-contracting distortion d embedding f_{len} of $C \cup S_i$ into a path $P_{\mathsf{len}} = \{1, 2, \ldots, \mathsf{len}\}$ such that f_{len} restricted to the first $|S_i|$ vertices is same as the mapping by Ψ, and for each $u \in C \cup S_i$, $D_{P_{\mathsf{len}}}(1, f_{\mathsf{len}}(u)) \leq D_{P_{\mathsf{len}}}(1, f_{\mathsf{len}}(\ell))$. Such an algorithm is described in the full version of the paper. If the algorithm returns no for all lengths, for every candidate ℓ for the last vertex, then there does not exist any special embedding of G into H with respect to $\mathcal{Y}$. Otherwise, assume that for the current guess ℓ, f_C is an embedding that the algorithm returns for the shortest length.

Let $F = \Phi_\Pi$ be the function such that $\Pi = \{\Psi\} \cup \{f_C \mid C \text{ is a component of } G \setminus U\}$. We verify whether the obtained F is a non-contracting distortion d embedding from G to H. If yes, we are done. If not, then there does not exist any special embedding with respect to $\mathcal{Y}$. Observe that the distance between two given points in H, can be computed in $O(1)$ time using $\mathcal{D}_{st}$.

Note that in the worst case, we have to run the above steps for all possible configurations. If we decide that there does not exist a special embedding with respect to all configurations, then we report that G does not admit the desired embedding of G into H. The correctness of the algorithm follows from Lemma 3.14. ◀

4 Graph Metric Embedding and connected treewidth

In this Section, we will look at the Graph Metric Embedding problem with respect to the added parameters of treewidth and longest geodesic cycle of the output graph metric. Let (G, D_G) be the input connected graph metric to be embedded into (H, D_H). We show that this problem is FPT, when parameterized by the distortion d, the treewidth $tw(H) = \alpha$, the length ℓ_g of the longest geodesic cycle of H, and the maximum degree $\Delta(H) = \Delta$. From [6] it can be shown that for a graph with longest geodesic cycle ℓ_g, a tree decomposition of treewidth α' can be converted into a connected tree decomposition of width $\alpha' + \binom{\alpha'}{2}(\ell_g(\alpha' - 2) - 1)$ in polynomial time. Since trees have constant connected treewidth, our algorithm is a generalization of the FPT algorithm for Graph Metric Embedding for trees, parameterized by distortion d and maximum degree Δ [7]. As before, we employ a dynamic programming to build a non-contracting distortion d metric embedding using a set of partial embeddings that are computed in FPT time.

Before we give the details of the algorithm, we want to make the following remark about bijective Graph Metric Embedding. We extended the algorithm of Kenyon et al [12] for bijective embedding of unweighted graphs into bounded maximum degree trees to the case of graphs with bounded maximum degree and bounded treewidth (see Theorem 1.2). The techniques we use for the results in this section are a generalization of the techniques used to prove Theorem 1.2. For the details of the proof, please refer to the full version of the paper [8].

Let (G, D_G) be a graph metric to be embedded into (H, D_H). Here the parameters are the treewidth α of H, the length of the longest geodesic cycle ℓ_g in H, the distortion d and the maximum degree Δ of H. Let $\mathcal{T}$ be a nice tree decomposition of H with width μ. Since from [6] H has a connected tree decomposition of width μ, we may assume that the nice tree decomposition is derived from the connected tree decomposition [5] and therefore the maximum distance between any two vertices inside a bag in $\mathcal{T}$ is $\Gamma \leq \mu$.

Ensuring non-contraction for a non-contracting distortion d metric embedding F is more elaborate. Local non-contraction no longer implies global non-contraction. This problem was dealt with in [7] by introducing the notion of *types*. For our algorithm too, for a vertex $\mathbf{u} \in V(T)$ we need to define a *type* for every vertex of $V(G)$ that is mapped into the subgraph $H_{\mathbf{u}}$, to indicate how it behaves with the rest of the graph. Informally, the types store information of the interaction of vertices of the graph seen so far with the boundary vertices, and this is enough to ensure global non-contraction.

▶ **Definition 4.1.** Let $\mathbf{u} \in V(T)$, $f_{\mathbf{u}}$ be a feasible partial embedding and $X_{\mathbf{u}} = \{u_1, \ldots, u_k\}$, $1 \leq k \leq \alpha_c$. Then:

(i) For $\mathbf{v} \in N_T(\mathbf{u})$ and $u_i \in X_{\mathbf{u}}$, $[f_{\mathbf{u}}, \mathbf{v}, u_i]$ *type* is a function $t^{u_i} : \mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{v}) \to \{\infty, 2\Gamma + 3d + 3, \Gamma + d + 1, \ldots, -(\Gamma + d + 1)\}$,

(ii) A $[f_{\mathbf{u}}, \mathbf{v}]$ *type* $\mathbf{t}$ is a tuple $(t^{u_i}, \ldots, t^{u_k})$, where t^{u_i} is a $[f_{\mathbf{u}}, \mathbf{v}, u_i]$ type, and

(iii) A $[f_{\mathbf{u}}, \mathbf{v}]$ *type-list* is a set of $[f_{\mathbf{u}}, \mathbf{v}]$ types.

Intuitively, we want to define a type corresponding to each vertex mapped into $H_{\mathbf{u}}$. However, this blows up the number of types. In order to handle this, it can be shown that we do not need to remember the type of each vertex, and that it is enough to only remember the type of vertices "close to" the vertices in $X_{\mathbf{u}}$. Now we present the formal arguments. To bound the total number of possible types, we define a function β as follows: $\beta(k) = k$ if $k < 2\Gamma + 3d + 3$, and $\beta(k) = \infty$ otherwise. In the following definitions, treat $\beta(k) = k$ and the definition of β will be clear while we prove our claims.

▶ **Definition 4.2.** Let us consider $\mathbf{u} \in V(T)$, $\mathbf{v} \in N_T(\mathbf{u})$. Let $f_{\mathbf{u}}$ be a feasible partial embedding and $\mathcal{L}$ be a $[f_{\mathbf{u}}, \mathbf{v}]$ type-list. Then $\mathcal{L}$ is said to be *compatible* with $\mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{v})$ if the following condition is satisfied: For each $x \in \mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{v})$ there exists a type $\mathbf{t} \in \mathcal{L}$, such that for each $y \in \mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{v})$, for all $u_i \in X_{\mathbf{u}}$ $D_H(f_{\mathbf{u}}(x), u_i) - D_G(x, y) = t^{u_i}(y)$.

▶ **Definition 4.3.** Let $\mathbf{u} \in V(T)$ and $f_{\mathbf{u}}$ be a feasible partial embedding. Also consider $\mathbf{v}, \mathbf{w} \in N_T(\mathbf{u})$ along with a $[f_{\mathbf{u}}, \mathbf{v}]$ type-list $\mathcal{L}_1$ and a $[f_{\mathbf{u}}, \mathbf{w}]$ type-list $\mathcal{L}_2$ such that $\mathbf{v} \neq \mathbf{w}$. Then $\mathcal{L}_1$ and $\mathcal{L}_2$ *agree* if the following condition is satisfied for all $u_i \in X_{\mathbf{u}}$: For every $\mathbf{t}_1 \in \mathcal{L}_1$ and $\mathbf{t}_2 \in \mathcal{L}_2$, there exists $x \in \mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{v})$ and $y \in \mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{w})$ such that $t_1^{u_i}(x) + t_2^{u_i}(y) \geq D_G(x, y)$ for all $u_i \in X_{\mathbf{u}}$.

Next, we define a state with respect to a vertex in T.

▶ **Definition 4.4.** Let $\mathbf{u} \in V(T)$. A **u**-*state* constitutes of a feasible partial embedding $f_{\mathbf{u}}$, a $[f_{\mathbf{u}}, \mathbf{v}]$ type-list $\mathcal{L}[f_{\mathbf{u}}, v]$ for each $\mathbf{v} \in N_T(\mathbf{u})$.

Notice that it is no longer enough to consider feasibility and succession of partial embeddings. We also need to take care of the types of vertices. Therefore, we define feasibility and succession of states.

▶ **Definition 4.5.** A **u**-state is said to be *feasible* if the following conditions are satisfied:

(i) $\mathcal{L}[f_{\mathbf{u}}, \mathbf{v}]$ is compatible with $\mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{v})$, for each $\mathbf{v} \in N_T(\mathbf{u})$, and

(ii) $\mathcal{L}[f_{\mathbf{u}}, \mathbf{v}]$ agrees with $\mathcal{L}[f_{\mathbf{u}}, \mathbf{w}]$, for any $\mathbf{v}, \mathbf{w} \in N_T(\mathbf{u})$ and $\mathbf{v} \neq \mathbf{w}$.

▶ **Definition 4.6.** Let $\mathbf{u} \in V(T)$ and $\mathbf{v} \in C_T(\mathbf{u})$. Let $\mathcal{S}_{\mathbf{u}}, \mathcal{S}_{\mathbf{v}}$ be feasible **u**-state and **v**-state, respectively. $\mathcal{S}_{\mathbf{v}}$ is said to *succeed* $\mathcal{S}_{\mathbf{u}}$ if the following properties hold.

(i) $f_{\mathbf{v}}$ succeeds $f_{\mathbf{u}}$.

(ii) For every $\mathbf{w} \in N_T(\mathbf{v}) \setminus \mathbf{u}$ and a type $\mathbf{t}_1 \in \mathcal{L}[f_{\mathbf{v}}, \mathbf{w}]$ there exists a type $\mathbf{t}_2 \in \mathcal{L}[f_{\mathbf{u}}, \mathbf{v}]$ satisfying the following conditions: **(a)** $\forall x \in \mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{v}) \cap \mathsf{Dom}_{f_{\mathbf{v}}}(\mathbf{w})$ and $a \in X_{\mathbf{u}} \cap X_{\mathbf{v}}$, $t_2^a(x) = t_1^a(x)$. **(b)** $\forall x \in \mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{v}) \cap \mathsf{Dom}_{f_{\mathbf{v}}}(\mathbf{w})$ and $a \in X_{\mathbf{u}} \setminus X_{\mathbf{v}}$, $t_2^a(x) = \beta(\min_{b \in X_{\mathbf{v}}} (D_H(a, b) + t_1^b(x)))$. **(c)** $\forall x \in \mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{v}) \setminus \mathsf{Dom}_{f_{\mathbf{v}}}(\mathbf{w})$ and $a \in X_{\mathbf{u}} \cap X_{\mathbf{v}}$, $t_2^a(x) = \beta(\max_{y \in \mathsf{Dom}_{f_{\mathbf{v}}}(\mathbf{w})} (t_1^a(y)) - D_G(x, y)))$. **(d)** $\forall x \in \mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{v}) \setminus \mathsf{Dom}_{f_{\mathbf{v}}}(\mathbf{w})$ and $a \in X_{\mathbf{u}} \setminus X_{\mathbf{v}}$, $t_2^a(x) = \beta(\max_{y \in \mathsf{Dom}_{f_{\mathbf{v}}}(\mathbf{w})} (\min_{b \in X_{\mathbf{v}}} (D_H(a, b) + t_1^b(y)) - D_G(x, y)))$.

(iii) For every $\mathbf{w} \in N_T(\mathbf{u}) \setminus \mathbf{v}$ and a type $\mathbf{t}_1 \in \mathcal{L}[f_{\mathbf{u}}, \mathbf{w}]$ there exists a type $\mathbf{t}_2 \in \mathcal{L}[f_{\mathbf{v}}, \mathbf{u}]$ satisfying the following conditions: **(a)** $\forall x \in \mathsf{Dom}_{f_{\mathbf{v}}}(\mathbf{u}) \cap \mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{w})$ and $a \in X_{\mathbf{u}} \cap X_{\mathbf{v}}$, $t_2^a(x) = t_1^a(x)$. **(b)** $\forall x \in \mathsf{Dom}_{f_{\mathbf{v}}}(\mathbf{u}) \cap \mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{w})$ and $a \in X_{\mathbf{v}} \setminus X_{\mathbf{u}}$, $t_2^a(x) = \beta \min_{b \in X_{\mathbf{u}}} (D_H(a, b) +$

$t_1^b(x))$. **(c)** $\forall x \in \mathsf{Dom}_{f_{\mathbf{v}}}(\mathbf{u}) \setminus \mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{w})$ and $a \in X_{\mathbf{u}} \cap X_{\mathbf{v}}$, $t_2^a(x) = t_1^a(x)$. **(d)** $\forall x \in \mathsf{Dom}_{f_{\mathbf{v}}}(\mathbf{u}) \setminus \mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{w})$ and $a \in X_{\mathbf{v}} \setminus X_{\mathbf{u}}$, $t_2^a(x) = \beta(\max_{y \in \mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{w})}(\min_{b \in X_{\mathbf{u}}}(D_H(a,b) + t_1^b(y)) - D_G(x,y)))$.

Now, we define the embeddability of a set of feasible states.

▶ **Definition 4.7.** For $u \in V(T)$, let $\mathcal{S}_u$ denote a u-state. The set $\{\mathcal{S}_{\mathbf{u}} : \mathbf{u} \in V(T)\}$ is said to be an *embeddable* set of feasible states if the following conditions are satisfied: (i) For each $\mathbf{u} \in V(T)$, $\mathcal{S}_{\mathbf{u}}$ is a feasible state, and (ii) For $\mathbf{u} \in V(T)$ and $\mathbf{v} \in C_T(\mathbf{u})$, $\mathcal{S}_{\mathbf{v}}$ succeeds $\mathcal{S}_{\mathbf{u}}$.

The above definitions are enough to show the relation between the existence of a non-contracting distortion d embedding of G into H and the existence of an embeddable set of feasible states. This is proved over the following two Lemmas. Lemma 4.9 is the most important structural Lemma for the design of this algorithm. We give a brief sketch of this Lemma and refer to the full details in the full version. For the proof of Lemma 4.8 refer to the full version.

▶ **Lemma 4.8.** *Let F be a non-contracting and distortion d embedding of G into H. Then there exists an embeddable set of feasible states.*

▶ **Lemma 4.9.** *Let $\Pi = \{f_{\mathbf{u}} : \mathbf{u} \in V(T)\}$ be an embeddable set of feasible states. Then there exists a non-contracting and distortion d embedding of G into H.*

Proof Sketch. To prove this lemma we first show the following:

1. For every $x \in V(G)$, there exists a feasible u-state such that $x \in \mathsf{Dom}_{f_{\mathbf{u}}}$, and
2. The subgraph of T induced by $A_x = \{\mathbf{u} \in V(T) : x \in \mathsf{Dom}_{f_{\mathbf{u}}}\}$ is connected. Moreover, $x \in \mathsf{Dom}_{f_{\mathbf{u}}} \cap \mathsf{Dom}_{f_{\mathbf{v}}}$ implies $f_{\mathbf{u}}(x) = f_{\mathbf{v}}(x)$.

Next, using the family Π, we construct an embedding F that satisfies the following (Please refer to the full version for this construction):

(i) F is a metric embedding with expansion at most d,

(ii) Consider a path $P = \mathbf{u}_1\mathbf{u}_2 \dots \mathbf{u}_k$ from $\mathbf{u} = \mathbf{u}_1$ to $\mathbf{v} = \mathbf{v}_k$ in T. Then for every $x \in \mathsf{Dom}_{f_{\mathbf{v}}}$, at least one of the following properties hold.

Prop-1: There exists a $\mathbf{u}_j \in P$ and $y \in \mathsf{Dom}_{f_{\mathbf{u}_j}}$ such that $D_H(F(x), u') - D_G(x,y) \geq 2\Gamma + 3d + 3$ for all $u' \in X_{\mathbf{u}_j}$.

Prop-2: There exists a type $\mathbf{t}_x \in \mathcal{L}[f_{\mathbf{u}}, \mathbf{u}_2]$ such that $\mathbf{t}_x^{u'}(y) = D_H(F(x), u') - D_G(x,y)$ for all $y \in \mathsf{Dom}_{f_{\mathbf{u}}}(\mathbf{u}_2)$ and $u' \in X_{\mathbf{u}}$.

(iii) Consider a path $P = \mathbf{u}_1\mathbf{u}_2 \dots \mathbf{u}_k$ from $\mathbf{u}$ to $\mathbf{v}$ in T, where $\mathbf{u} = \mathbf{u}_1$ and $\mathbf{v} = \mathbf{u}_k$. Then F restricted to $\bigcup_{\mathbf{u}_i \in P} \mathsf{Dom}_{f_{\mathbf{u}_i}}$ is non-contracting.

Now we will be done if we prove that F is a non-contracting embedding for any two vertices $x, y \in V(G)$. Note that each of $F(x)$ and $F(y)$ is in some bag. Fix $\mathbf{u}, \mathbf{v} \in V(T)$ such that $F(x) \in X_{\mathbf{u}}$ and $F(y) \in X_{\mathbf{v}}$. Consider the path $P = \mathbf{u}_1\mathbf{u}_2 \dots \mathbf{u}_k$ from $\mathbf{u}$ to $\mathbf{v}$ in T, where $\mathbf{u} = \mathbf{u}_1$ and $\mathbf{v} = \mathbf{u}_k$. We can show that the shortest path between $F(x)$ to $F(y)$ is non-contracting as $x, y \in \bigcup_{\mathbf{u}_i \in P} \mathsf{Dom}_{f_{\mathbf{u}_i}}$. ◀

Proof Ideas for Theorem 1.3. A graph that is embeddable into the given H must have bounded maximum degree. This helps in proving bounds for the total number of feasible partial embeddings and the total number of feasible states. After this, the proof of Theorem 1.3 uses the standard dynamic programming approach over a bounded tree-decomposition of a graph. ◀

5 Open Questions

The parameterized complexity of embedding into trees of unbounded degree, asked in [7], still remains open. A generalization of that question is to determine the parameterized complexity of Graph Metric Embedding for bounded treewidth graphs, and this is also open.

References

1 I. Abraham, Y. Bartal, and O. Neiman. Advances in metric embedding theory. *Advances in Mathematics*, 228(6):3026–3126, 2011.
2 M. Badoiu, J. Chuzhoy, P. Indyk, and A. Sidiropoulos. Low-Distortion Embeddings of General Metrics into the Line. In *Proc. of STOC*, pages 225–233, 2005.
3 M. Badoiu, K. Dhamdhere, A. Gupta, Y. Rabinovich, H. Räcke, R. Ravi, and A. Sidiropoulos. Approximation Algorithms for Low-Distortion Embeddings into Low-Dimensional spaces. In *Proc. of SODA*, pages 119–128, 2005.
4 T. Carpenter, F. V. Fomin, D. Lokshtanov, S. Saurabh, and A. Sidiropoulos. Algorithms for low-distortion embeddings into arbitrary 1-dimensional spaces. *CoRR*, abs/1712.06747, 2017. To appear in *SoCG*, 2018.
5 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. Parameterized algorithms. *Springer*, 5(4):16:1–16:20, 2015.
6 Reinhard Diestel and Malte Müller. Connected tree-width. *Combinatorica*, 2017.
7 M. R. Fellows, F. V. Fomin, D. Lokshtanov, E. Losievskaja, F. A. Rosamond, and S. Saurabh. Distortion is Fixed Parameter Tractable. *ACM Transactions on Computation Theory*, 5(4):16:1–16:20, 2013.
8 A. Ghosh, S. Kolay, and G. Mishra. FPT algorithms for embedding into low complexity graphic metrics. *CoRR*, 2018. `arXiv:1801.03253`.
9 A. Gupta, I. Newman, Y. Rabinovich, and A. Sinclair. Cuts, Trees and l_1-Embeddings of Graphs. *Combinatorica*, 24(2):233–269, 2004.
10 P. Indyk. Algorithmic Applications of Low-Distortion Geometric Embeddings. In *Proc. of FOCS 2001*, pages 10–33, 2001.
11 P. Indyk and J. Matoušek. Low-Distortion Embeddings of Finite Metric Spaces. In *Handbook of Discrete and Computational Geometry, Second Edition.*, pages 177–196. CRC, 2004.
12 C. Kenyon, Y. Rabani, and A. Sinclair. Low Distortion Maps Between Point Sets. *SIAM Journal on Computing*, 39(4):1617–1636, 2010. Peliminary version in *Proc. of STOC*, 2004.
13 N. Linial, E. London, and Y. Rabinovich. The Geometry of Graphs and Some of its Algorithmic Applications. *Combinatorica*, 15(2):215–245, 1995.
14 J. Matoušek. *Lectures on Discrete Geometry*. Springer-Verlag New York, Inc., 2002.
15 J. Matoušek. Lecture Notes on Metric Embeddings, 2013.
16 A. Nayyeri and B. Raichel. Reality Distortion: Exact and Approximate Algorithms for Embedding into the Line. In *Proc. of FOCS*, pages 729–747, 2015.
17 A. Nayyeri and B. Raichel. A Treehouse with Custom Windows: Minimum Distortion Embeddings into Bounded Treewidth Graphs. In *Proc. of SODA*, pages 724–736, 2017.

The Stochastic Score Classification Problem

Dimitrios Gkenosis
Department of Informatics and Telecommunications, University of Athens, Athens, Greece
gkenosis.dimitrios@math.uoa.gr

Nathaniel Grammel
Department of Computer Science, University of Maryland, College Park, Maryland, USA
ngrammel@cs.umd.edu

Lisa Hellerstein
Department of Computer Science and Engineering, NYU Tandon School of Engineering, Brooklyn, NY, USA
lisa.hellerstein@nyu.edu

Devorah Kletenik[1]
Department of Computer and Information Science, Brooklyn College, CUNY, Brooklyn, New York, USA
kletenik@sci.brooklyn.cuny.edu

Abstract

Consider the following Stochastic Score Classification Problem. A doctor is assessing a patient's risk of developing a certain disease, and can perform n tests on the patient. Each test has a binary outcome, positive or negative. A positive result is an indication of risk, and a patient's score is the total number of positive test results. Test results are accurate. The doctor needs to classify the patient into one of B risk classes, depending on the score (e.g., LOW, MEDIUM, and HIGH risk). Each of these classes corresponds to a contiguous range of scores. Test i has probability p_i of being positive, and it costs c_i to perform. To reduce costs, instead of performing all tests, the doctor will perform them sequentially and stop testing when it is possible to determine the patient's risk category. The problem is to determine the order in which the doctor should perform the tests, so as to minimize expected testing cost. We provide approximation algorithms for adaptive and non-adaptive versions of this problem, and pose a number of open questions.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases approximation algorithms, symmetric Boolean functions, stochastic probing, sequential testing, adaptivity

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.36

Related Version Full version on arXiv [9], `https://arxiv.org/abs/1806.10660`.

Funding The authors were partially supported by NSF Award IIS-1217968.

Acknowledgements We thank Zach Pomerantz for experiments that gave us useful insights into goal functions. We thank the ESA referees for helpful comments.

[1] Partially supported by a PSC-CUNY Award, jointly funded by The Professional Staff Congress and The City University of New York

© Dimitrios Gkenosis, Nathaniel Grammel, Lisa Hellerstein, and Devorah Kletenik;
licensed under Creative Commons License CC-BY

1 Introduction

We consider the following *Stochastic Score Classification (SSClass)* problem. A doctor wants to assess a patient's risk of developing a certain disease, and can perform n tests on the patient. Each test has a binary outcome, positive or negative. A positive result is an indication of risk, and a patient's score is the total number of positive test results. Test results are accurate. The doctor needs to classify the patient into one of B risk classes, depending on the score (e.g., LOW, MEDIUM, and HIGH risk). Each of these classes corresponds to a contiguous range of scores. Test i has probability p_i of being positive, and it costs c_i to perform. To reduce costs, instead of performing all tests, the doctor will perform them sequentially and stop testing when it is possible to determine the risk category for the patient.

To reduce costs, instead of performing all tests and computing an exact score, the doctor will perform them sequentially, stopping when the class becomes a foregone conclusion. For example, suppose there are 10 tests and the MEDIUM class corresponds to a score between 4 and 7 inclusive. If the doctor performed 8 tests, of which 5 were positive, the doctor would not perform the remaining 2 tests, because the patient's risk class will be MEDIUM regardless of the outcome of the 2 remaining tests. The problem is to determine the optimal (adaptive or non-adaptive) order in which to perform the tests, so as to minimize expected cost.

Formally, the Stochastic Score Classification problem is as follows. Given $B+1$ integers $0 = \alpha_1 < \alpha_2 < \ldots < \alpha_B < \alpha_{B+1} = n+1$, let *class* j correspond to the *scoring interval* $[\alpha_j, \alpha_j+1, \ldots, \alpha_{j+1}-1]$. The α_j define an associated pseudo-Boolean *score classification function* $f : \{0,1\}^n \rightarrow \{1,\ldots,B\}$, such that $f(X_1,\ldots,X_n)$ is the class whose scoring interval contains the *score* $\sum_i X_i$. Thus B is the number of classes. Each variable X_i is independently 1 with given probability p_i, where $0 < p_i < 1$, and is 0 otherwise. The value of X_i can only be determined by asking a query (or performing a test), which incurs a given positive, real-valued cost c_i.

An *evaluation strategy* for f is a sequential adaptive or non-adaptive order in which to ask the queries. Querying must continue until the value of f can be determined, i.e., until the value of f would be the same, no matter how the remainder of the n queries were answered. In an *adaptive evaluation strategy*, the choice of the next query can depend on the outcomes of previous queries. An adaptive strategy corresponds to a decision tree, although we do not require the tree to be output explicitly (it may have exponential size). A *non-adaptive strategy* is a permutation of the queries. With a non-adaptive strategy, querying proceeds in the order specified by the permutation until the value of f can be determined.

Repeated queries always receive the same response, so it is never useful to ask a particular query more than once. The goal is to design an evaluation strategy for f with minimum expected total query cost. We consider both adaptive and non-adaptive versions of the problem, in which we are restricted to adaptive or non-adaptive strategies respectively.

We also consider a *weighted* variant of the problem, where query i has given integer weight a_i, the score is $\sum_i a_i X_i$, and $\alpha_1 < \alpha_2 < \ldots < \alpha_B < \alpha_{B+1}$ where α_1 equals the minimum possible value of the score $\sum_i a_i X_i$, and $\alpha_{B+1} - 1$ equals the maximum possible value. We refer to the standard version of the problem, with score $\sum_i X_i$, as the *unweighted* version.

While we have described the problem above in the context of assessing disease risk, score classification is also used in other contexts, such as assigning letter grades to students, giving a quality rating to a product, or deciding whether a person charged with a crime should be released on bail. In Machine Learning, the focus is on learning the score classification function [25, 23, 15, 27, 26]. In contrast, here our focus is on reducing the cost of evaluating the classification function. We note that the SSClass problem differs from many other stochastic probing problems previously considered (e.g. [22, 14]) because of the requirement that testing must continue until the unique interval containing the score has been determined.

Restricted versions of the weighted and unweighted SSClass problem have been studied previously. In the algorithms literature, Deshpande et al. presented two approximation algorithms solving the *Stochastic Boolean Function Evaluation* (SBFE) problem for linear threshold functions [8]. The general SBFE problem is similar to the adaptive SSClass problem, but instead of evaluating a given score classification function f defined by inputs α_j, you need to evaluate a given Boolean function f. The SBFE problem for linear threshold functions is equivalent to the weighted adaptive SSClass problem. One of the two algorithms of Deshpande et al. achieves an $O(\log W)$-approximation factor for this problem using the submodular goal value approach; it involves construction of a goal utility function and application of the Adaptive Greedy algorithm of Golovin and Krause to that function [10]. Here W is the sum of the magnitudes of the integer weights a_i. The other algorithm achieves a 3-approximation by applying a dual greedy algorithm to the same goal utility function.

A k-of-n function is a Boolean function f such that $f(x) = 1$ iff $x_1 + \ldots + x_n \geq k$. The SBFE problem for evaluating k-of-n functions is equivalent to the unweighted adaptive SSClass problem, with only two classes ($B = 2$). It has been studied previously in the VLSI testing literature. There is an elegant algorithm for the problem that computes an optimal strategy [19, 4, 20, 6].

The unweighted adaptive SSClass problem for arbitrary numbers of classes was studied in the information theory literature [7, 1, 17], but only for unit costs. The main novel contribution there was to establish an equivalence between verification and evaluation, which we discuss below.

2 Results and open questions

We give approximation results for adaptive and non-adaptive versions of the SSClass problem. We describe most of our results here, but leave description of some others to the full version of our paper [9]. Omitted proofs also appear there. A table with all our bounds can be found at the end of this paper.

We begin by using the submodular goal value approach of Deshpande et al. to obtain an $O(\log W)$ approximation algorithm for the weighted adaptive SSClass problem. This immediately gives an $O(\log n)$ approximation for the unweighted adaptive problem. We also present a simple alternative algorithm achieving a $B - 1$ approximation for the unweighted adaptive problem, and a $3(B-1)$-approximation algorithm for the weighted adaptive problem, again using an algorithm of Deshpande et al.

We then present our two main results, which are both for the case of unit costs. The first is a 4-approximation algorithm for the adaptive and non-adaptive versions of the unweighted SSClass problem. The second is a φ-approximation for a special case of the non-adaptive unweighted version, where the problem is to evaluate what we call the *Unanimous Vote* function. Here $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$ is the golden ratio. The Unanimous Vote function outputs POSITIVE if $X_1 = \ldots = X_n = 1$, NEGATIVE if $X_1 = \ldots = X_n = 0$, and UNCERTAIN otherwise. Equivalently, it is a score classification function with $B = 3$ and scoring intervals $\{0\}, \{1, \ldots, n-1\}$ and $\{n\}$. The proofs of our two main results imply upper bounds of 4 and φ for the adaptivity gaps of the corresponding problems.

We use both existing techniques and new ideas in our algorithms. We use the submodular goal value approach of Deshpande et al. to get our $O(\log W)$ bound for the weighted adaptive SSClass problem. This approach cannot yield a bound better than $O(\log n)$ for SSClass problems, since they involve evaluating a function of n relevant Boolean variables [3].

For some of our other bounds, we exploit the exact algorithm for k-of-n evaluation, and the ideas used in its analysis. To obtain non-adaptive algorithms for the unit-cost case, we perform a round robin between 2 subroutines, one performing queries in increasing order of c_i/p_i, while the second performs them in increasing order of $c_i/(1-p_i)$. For arbitrary costs, instead of standard round robin, we use the modified round robin approach of Allen et al [2]. As has been repeatedly shown, the c_i/p_i ordering and the $c_i/(1-p_i)$ ordering are optimal for evaluation of the Boolean OR (1-of-n) and AND (n-of-n) functions respectively (cf. [24]). Intuitively, the first ordering (for OR) favors queries with low cost and high probability of producing the value 1, while the second (for AND) favors queries with low cost and high probability of producing the value 0. The proof of optimality follows from the fact that given any ordering, swapping two adjacent queries that do not follow the designated increasing order will decrease expected evaluation cost.

While the algorithm for our first main result is very simple, the proof of its 4-approximation bound is not. It uses ideas from the existing analysis of the k-of-n algorithm, but that analysis is simpler because $B = 2$. We perform a new, careful analysis to obtain our 4-approximation result. Unlike the analysis of the k-of-n algorithm, our analysis only works for unit costs.

To develop our φ-approximation for the Unanimous Vote function, we first note that for such a function, if you perform the first query and observe its outcome, the optimal ordering of the remaining queries can be determined by evaluating a Boolean OR function, or a Boolean AND function. We then address the problem of determining an approximately optimal permutation, given the first query. A standard round robin alternating between the $c_i/p_i = 1/p_i$ ordering, and the $1/(1-p_i)$ ordering, yields a factor of 2 approximation. To obtain the φ factor, we stop the round robin at a carefully chosen point and *commit* to one of the two orderings, abandoning the other. Our full algorithm for the Unanimous Vote function works by trying all n possible first queries. For each, we generate the approximately optimal permutation given that first choice, and algebraically compute its expected cost. Finally, out of these n permutations, we choose the one with lowest expected cost.

We note that although our algorithms are designed to minimize expected cost for independent queries, the goal value function used to achieve the $O(\log W)$ approximation result can also be used to achieve a worst-case bound, and a related bound in the Scenario model [10, 12, 16].

A recurring theme in work on SSClass problems has been the relationship between the evaluation problems and their associated verification problems. In the verification problem, you are given the output class (i.e., the value of the score classification function) before querying, and just need to perform enough tests to verify that the given output class is correct. Thus optimal expected verification cost lower bounds optimal expected evaluation cost. Surprisingly, the result of Das et al. [7] showed that for the adaptive SSClass problem in the unit-cost case, optimal expected verification cost equals optimal expected evaluation cost. Prior work already implied this was true for evaluating k-of-n functions, even for arbitrary costs (cf. [5]). We give a counterexample in the full paper [9] showing that this relationship does not hold for the adaptive SSClass problem with arbitrary costs. Thus algorithmic approaches based on optimal verification strategies may not be effective for this problem.

There remain many intriguing open questions related to SSClass problems. The first, and most fundamental, is whether the (adaptive or non-adaptive) SSClass problem is NP-hard.

This is open even in the unit-cost case. It is unclear whether this problem will be easy to resolve. It is easy to show that the weighted variants are NP-hard: this follows from the NP-hardness of the SBFE problem for linear threshold functions, which is proved by a simple reduction from knapsack [8]. However, the approach used in that proof is to show that the deterministic version of the problem (where query answers are known a-priori) is NP-hard, which is not the case in the SSClass problem. Further, NP-hardness of evaluation problems is not always easy to determine. The question of whether the SBFE problem for read-once formulas is NP-hard has been open since the 1970's (cf. [13]).

Another main open question is whether there is a constant-factor approximation algorithm for the weighted SSClass problem. Our bounds depend on n or B. Other open questions concern lower bounds on approximation factors, and bounds on adaptivity gaps.

3 Further definitions and background

A *partial assignment* is a vector $b \in \{0, 1, *\}^n$. We use f^b to denote the restriction of function $f(x_1, \ldots, x_n)$ to the bits i with $b_i = *$, produced by fixing the remaining bits i according to their values b_i. We call f^b the function *induced from f by partial assignment b*. We use $N_0(b)$ to denote $|\{i|b_i = 0\}|$, and $N_1(b)$ to denote $|\{i|b_i = 1\}|$.

A partial assignment $b' \in \{0, 1, *\}^n$ is an *extension* of b, written $b' \succeq b$, if $b'_i = b_i$ for all i such that $b_i \neq *$. We use $b' \succ b$ to denote that $b' \succeq b$ and $b' \neq b$.

A partial assignment encodes what information is known at a given point in a sequential querying (testing) environment. Specifically, for partial assignment $b \in \{0, 1, *\}^n$, $b_i = *$ indicates that query i has not yet been asked, otherwise b_i equals the answer to query i. We may also refer to query i as *test* i, and to asking query i as testing or querying bit x_i,

Suppose the costs c_i and probabilities p_i for the n queries are fixed. We define the expected costs of adaptive evaluation and verification strategies for $f\colon \{0, 1\}^n \to \{0, 1\}$ or $f\colon \{0, 1\}^n \to \{1, \ldots, B\}$ as follows. (The definitions for non-adaptive strategies are analogous.) Given an adaptive evaluation strategy $\mathcal{A}$ for f, and an assignment $x \in \{0, 1\}^n$, we use $C(\mathcal{A}, x)$ to denote the sum of the costs of the tests performed in using $\mathcal{A}$ on x. The expected cost of $\mathcal{A}$ is $\sum_{x \in \{0,1\}^n} C(\mathcal{A}, x)p(x)$, where $p(x) = \prod_{i=1}^{n} p^{x_i}(1-p)^{1-x_i}$. We say that $\mathcal{A}$ is an *optimal* adaptive evaluation strategy for f if it has minimum possible expected cost.

Let L denote the range of f, and for $\ell \in L$, let $\mathcal{X}_\ell = \{x \in \{0, 1\}^n : f(x) = \ell\}$. An *adaptive verification strategy* for f consists of $|L|$ adaptive evaluation strategies $\mathcal{A}_\ell$ for f, one for each $\ell \in L$. The expected cost of the verification strategy is $\sum_{\ell \in L} \left(\sum_{x \in \mathcal{X}_\ell} C(\mathcal{A}_\ell, x)p(x)\right)$ and it is optimal if it minimizes this expected cost.

If $\mathcal{A}$ is an evaluation strategy for f, we call $\sum_{x \in \mathcal{X}_\ell} C(\mathcal{A}, x)p(x)$ the ℓ-cost of $\mathcal{A}$. For $\ell \in L$, we say that $\mathcal{A}$ is *ℓ-optimal* if it has minimum possible ℓ-cost. In an optimal verification strategy for f, each component evaluation strategy $\mathcal{A}_\ell$ must be ℓ-optimal.

A function $g\colon \{0, 1, *\}^n \to \mathbb{Z}_{\geq 0}$ is *monotone* if $g(b') \geq g(b)$ whenever $b' \succeq b$. It is *submodular* if for $b' \succeq b$, i such that $b'_i = b_i = *$, and $k \in \{0, 1\}$, we have $g(b'_{i \leftarrow k}) - g(b') \leq g(b_{i \leftarrow k}) - g(b)$. Here $b_{i \leftarrow k}$ denotes the partial assignment produced from b by setting b_i to k, and similarly for $b'_{i \leftarrow k}$.

4 Algorithms for the weighted adaptive SSClass problem

Our first algorithm solves the weighted adaptive SSClass Problem using the *goal value approach* of Deshpande et al., a method of designing approximation algorithms for SBFE problems [8]. The approach can easily be extended to problems of evaluating pseudo-Boolean functions. It requires construction of a utility function $g\colon \{0,1,*\}^n \to \mathbb{Z}_{\geq 0}$, called a *goal function*, associated with the function f being evaluated. Function g must be monotone and submodular. The maximum value of g must be an integer $Q \geq 0$ such that $g(b) = Q$ iff $f(x)$ has the same value for all $x \in \{0,1\}^n$ such that $x \succeq b$. We call Q the *goal value* of g.

An adaptive strategy for evaluating f can then be obtained by applying the Adaptive Greedy algorithm of Golovin and Krause to solve the Stochastic Submodular Cover problem on goal function g [10]. This algorithm greedily chooses the test with highest expected increase in utility, as measured by g, per unit cost. It follows from the bound of Deshpande et al. on applying Adaptive Greedy to the Stochastic Submodular Cover problem, that this strategy is an $O(\log Q)$-approximation to the optimal adaptive strategy for evaluating f [8].[2]

We construct g as follows. Let $r(x) = a_1x_2 + \ldots + a_nx_n$. Consider an associated score classification function f defined by $\alpha_1, \ldots, \alpha_{B+1}$ and the a_i. For simplicity, we assume here that the a_i are non-negative. (The general case is similar.) We refer to the values $\alpha_2, \ldots, \alpha_B$ as *cutoffs*. For each cutoff α_j, let f_j denote the Boolean linear threshold function $f_j : \{0,1\}^n \to \{0,1\}$ where $f_j(x) = 1$ if $r(x) \geq \alpha_j$, and $f_j(x) = 0$ otherwise.

Consider a fixed cutoff α_j. Let $\omega = (\sum_i a_i) - \alpha_j + 1$. For $b \in \{0,1,*\}^n$, let $r^1(b) = \min\{\alpha_j, \sum_{i:b_i=1} a_i\}$ and $r^0(b) = \min\{\omega, \sum_{i:b_i=0} a_i\}$. Note that $r^1(b) = \alpha_j$ iff $f_j(x) = 1$ for all $x \succeq b$, and $r^0(b) = \omega$ iff $f_j(x) = 0$ for all $x \succeq b$. As shown in [8] the following function g_j is a goal function for linear threshold function f_j, with goal value $\omega\alpha_j$:

$$g_j(b) = \omega\alpha_j - (\alpha_j - r^1(b))(\omega - r^0(b)). \tag{1}$$

We combine the $B-1$ goal functions g_j using the standard "AND construction" for utility functions (cf. [8]), which yields a goal function g for score classification function f, where $g(x) = \sum_{i=1}^{B-1} g_i(x)$. Its goal value is at most $(B-1)W^2$ where $W = \sum_i a_i$.

To evaluate f, we apply the Adaptive Greedy algorithm to g. By the $O(\log Q)$ approximation bound on Adaptive Greedy, this constitutes an algorithm for the weighted adaptive SSClass problem with approximation factor $O(\log BW^2)$, which is $O(\log W)$ since $B \leq W$. In the unweighted adaptive SSClass problem, $W = n$, so the approximation factor is $O(\log n)$.

We now describe our simple $B-1$ approximation algorithm for the adaptive unweighted SSClass problem, which takes a very different approach. It runs the k-of-n function evaluation algorithm $B-1$ times, each time setting k to be a different cutoff α_j. The resulting evaluations are sufficient to determine the correct output class. The proof that this algorithm achieves a $B-1$ approximation bound is based on the observation that any strategy solving the adaptive SSClass problem is implicitly a strategy for solving each of the $B-1$ induced k-of-n problems. Since we use an optimal algorithm for solving each of those problems, this implies the $B-1$ approximation bound. We note that although we could easily modify this algorithm to use binary search, we do not know how to prove that it results in an approximation bound that is better than $B-1$.

When B is small, as for, e.g., k-of-n functions and the Unanimous Vote function, $B-1$ is a good approximation. Otherwise, the $O(\log n)$ approximation achieved with the goal value approach may be better.

By similar arguments, the following is a $3(B-1)$ approximation for the weighted adaptive problem. For each cutoff α_j, use the 3-approximation algorithm of Deshpande et al. to evaluate linear threshold function f_j.

[2] Golovin and Krause originally claimed an $O(\log Q)$ bound for Stochastic Submodular Cover [10], but the proof was recently found to have an error [18]. They have since posted a new proof with an $O(\log^2 Q)$ bound [11]. Deshpande et al. proved an $O(\log Q)$ bound using a different proof technique [8].

Combining the above results, we have the following theorem.

▶ **Theorem 1.** *There are two different polynomial-time approximation algorithms, achieving approximation factors of $O(\log W)$ and $3(B-1)$ respectively, for the weighted adaptive SSClass problem. There is a polynomial-time algorithm that achieves a $B-1$-approximation for the unweighted adaptive SSClass problem.*

5 Constant-factor approximations for unit-cost problems

We begin by reviewing relevant existing techniques.

5.1 Adaptive Evaluation of k-of-n Functions

An optimal adaptive strategy, when f is a k-of-n function, was given by Salloum, Ben-Dov, and Breuer [19, 4, 20, 6, 21]. The difficulty in finding an optimal strategy is that you do not know a-priori whether the value of f will be 1 or 0. If 1, then (ignoring cost) it seems it would be better to choose tests with high p_i, since you want to get k 1-answers. Similarly, if 0, it seems it would be better to choose tests with low p_i. The algorithm of Salloum et al. is based on showing that when f is a k-of-n function, a 1-optimal strategy is to test the bits in increasing order of c_i/p_i until getting k 1's, while a 0-optimal strategy is to test them in increasing order of $c_i/(1-p_i)$ until getting $n-k+1$ 0's.

Since the 1-optimal strategy must perform at least the first k tests before terminating, these can be reordered within this strategy without affecting its optimality. Similarly, the first $n-k+1$ queries of the 0-optimal strategy can be reordered without affecting optimality.

The strategy of Salloum et al. is as follows. If $n=1$, test the one bit. Else let S_1 denote the set of the k bits with smallest c_i/p_i values. Let S_0 denote the set of the $n-k+1$ bits with smallest $c_i/(1-p_i)$ values. Since $|S_0|+|S_1|=n+1$, by pigeonhole $S_0 \cap S_1 \neq \emptyset$. Test a bit in $S_0 \cap S_1$. If it is 1, the problem is reduced to evaluating the function $f^1\colon \{0,1\}^{n-1} \to \{0,1\}$ where $f^1(x)=1$ iff $N_1(x) \geq k-1$. If it is 0, the problem is reduced to evaluating $f^0\colon \{0,1\}^{n-1} \to \{0,1\}$ where $f^0(x)=1$ iff $N_1(x) \geq k$. Recursively evaluate f^1 or f^0 as appropriate. Optimality follows from the fact that the chosen bit is an optimal first bit to test in both 0-optimal and 1-optimal strategies.

5.2 Modified Round Robin

Allen et al. [2] presented a modified round robin protocol, which is useful in designing non-adaptive strategies when test costs are not all equal. Suppose that in a sequential testing environment with n tests, we have M conditions on test outcomes, corresponding to M predicates on the partial assignments in $\{0,1,*\}^n$. For example, in the k-of-n testing problem, we are interested in the following $M=2$ predicates on partial assignments: (1) having at least k 1's and (2) having at least $n-k+1$ 0's. Suppose we are given a testing strategy for each of the M predicates; a strategy stops testing when its predicate is satisfied (by the partial assignment representing test outcomes), or all tests have been performed. Let $\mathrm{Alg}_1, \ldots, \mathrm{Alg}_M$ denote those M strategies. The modified round robin algorithm of Allen et al. interleaves execution of these strategies. We present a version of their algorithm in Algorithm 1; the difference is that their algorithm terminates as soon as one of the predicates is satisfied, while Algorithm 1 terminates when all are satisfied.

Allen et al. showed that the modified round robin incurs a cost on x that is at most M times the cost incurred by Alg_j on x. We will use variations on this algorithm and this bound to derive approximation factors for our SSClass problems.

Algorithm 1 Modified Round Robin of M Strategies.

```
Let $C_i \leftarrow 0$ for $i = 1, \dots, M$; let $d \leftarrow (*^n)$
while at least one of the $M$ testing strategies has not terminated do
    Let $j_1, \dots, j_M$ be the next tests of $\mathrm{Alg}_1, \dots, \mathrm{Alg}_M$ respectively
    Let $i^* \leftarrow \arg\min_{i \in \{1,\dots,M\}} (C_i + c_{j_i})$
    Let $t \leftarrow j_{i^*}$; let $C_{i^*} \leftarrow C_{i^*} + c_t$
    Perform test $t$ and set $d_t$ to the newly determined value of bit $t$
end while
```

Algorithm 2 Non-adaptive Round Robin Algorithm for SSClass.

```
Let $C_0 \leftarrow 0$, $C_1 \leftarrow 0$
Let $d \leftarrow *^n$
repeat
    Let $j_0 \leftarrow$ next bit from $\mathrm{Alg}_0$
    Let $j_1 \leftarrow$ next bit from $\mathrm{Alg}_1$
    Let $j^* \leftarrow \arg\min_{i \in \{0,1\}} C_i + c_{j_i}$
    Query bit $i^*$ and set $d_{j^*}$ to the discovered value
until induced function $f^d$ is a constant function
return The constant value of $f^d$
```

5.3 A Round Robin Approach to Non-adaptive Evaluation

We now present an algorithm for the unit-cost case of the non-adaptive, unweighted SSClass problem. The pseudocode is presented in Algorithm 2, with Alg_1 denoting the strategy performing tests in increasing order of c_i/p_i and Alg_0 denoting the strategy performing tests in increasing order of $c_i/(1-p_i)$. We prove the following theorem.

▶ **Theorem 2.** *When all tests have unit cost, the expected cost incurred by the non-adaptive Algorithm 2 is at most 4 times the expected cost of an optimal adaptive strategy for the unweighted adaptive SSClass problem.*

By Theorem 2, Algorithm 2 is a 4-approximation for the adaptive *and* non-adaptive versions of the unit-cost unweighted SSClass problem. The theorem also implies an upper bound of 4 on the adaptivity gap for this problem. A simpler analysis shows that for arbitrary costs, Algorithm 2 achieves an approximation factor of $2(B-1)$ for the non-adaptive version of the problem. Since the k-of-n functions are essentially equivalent to score classification functions with $B = 2$, the $2(B-1)$-approximation is a 2-approximation for non-adaptive k-of-n function evaluation.

5.4 The Unanimous Vote Function: Adaptive Setting

Adaptive evaluation of the Unanimous Vote function can be done optimally using the following simple idea. Recall that querying the bits in increasing c_i/p_i order is optimal for evaluating OR, while querying in increasing $c_i/(1-p_i)$ is optimal for AND. Now consider the problem of adaptively evaluating the Unanimous Vote function. Suppose we know the optimal choice for the first test. After the first test, we have an induced SSClass problem on the remaining bits. If the first test has value 0, the induced function is equivalent to Boolean OR (mapping UNCERTAIN to 1, and NEGATIVE to 0). The subtree rooted at the root node's 0-child should be the optimal tree for evaluating OR. Specifically, the remaining bits should be tested

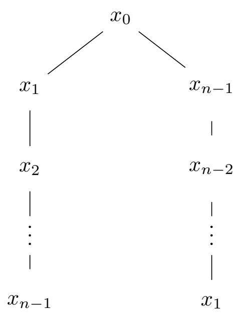


Figure 1 Decision tree T representing optimal adaptive strategy with root x_0.

in increasing order of c_i/p_i. If, instead, the first bit is 1, the induced function is equivalent to AND (mapping UNCERTAIN to 0 and POSITIVE to 1) and the remaining bits should be queried in increasing order of $c_i/(1-p_i)$.

Since we don't actually know the first bit, we can just try each bit as the root and build the rest of the tree according to the optimal OR and AND strategies. We can then calculate the expected cost of each tree, and output the tree with minimum expected cost.

For succinctness, the optimal OR and AND strategies can be represented by paths, because each performs tests in a fixed order. Figure 1 shows an example of the strategy computed by the algorithm, where the root is labeled x_0 and the OR permutation is the reversal of the AND permutation (which occurs, for example, with unit costs).

5.5 A Non-adaptive φ-approximation for the Unanimous Vote Function

A simple modification of the round robin makes the algorithm from the previous section non-adaptive, yielding a 2-approximation. But we now show how to achieve a non-adaptive φ-approximation in the unit-cost case, where $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$ is the golden ratio. We call the algorithm Truncated Round Robin. We describe the algorithm by describing a subroutine which generates a permutation of input bits to query, given an initial (root) bit. The algorithm then tries all possible bits for the root and chooses the resulting permutation that achieves the lowest expected cost.

Without loss of generality, assume the first bit (the root node) is x_0, and the rest are $x_1, \dots, x_{n-1}$, and $1 > p_1 \geq p_2 \geq \cdots \geq p_{n-1} > 0$. Fix c to be a constant such that $0 < c < \frac{1}{2}$.

The subroutine is shown in Algorithm 3. "Evaluation unknown" means tests so far were insufficient to determine the output of the Unanimous Vote function. (The output, POSITIVE, NEGATIVE, or UNCERTAIN, is not shown.)

Given x_0 as the root, the optimal adaptive strategy continues with the OR strategy (increasing $1/p_i$) when $x_0 = 0$, and the AND strategy (increasing $1/(1-p_i)$) when $x_0 = 1$. This is shown in Figure 1, where $x_0 = 0$ is the left branch and $x_0 = 1$ is the right. On the left, we stop querying when we find a bit with value 1 (or all bits are queried). On the right, we stop when we find a bit with value 0.

Let "level l" refer to the tree nodes at distance l from the root; namely, x_l and x_{n-l}. When all costs are 1, the standard round robin technique of the previous section in effect tests, for $l = 1 \dots \lceil \frac{n-1}{2} \rceil$, the bit x_l followed by x_{n-l}. Note that the algorithm will terminate by level $\lceil \frac{n-1}{2} \rceil$ because at this point all bits will have been queried. Thus in the algorithm, $p_l \geq p_{n-l}$.

Algorithm 3 Truncated Round Robin Subroutine for Unanimous Vote Fn.

```
Require: $1 > p_1 \geq p_2 \geq \cdots \geq p_{n-1}$
  Query bit $x_0$
  Let level $l \leftarrow 1$
  while $p_{n-l} < 1 - c$ and $p_l > c$ and evaluation unknown do
    if $|p_l - 0.5| < |p_{n-l} - 0.5|$ then
      Query $x_l$ followed by $x_{n-l}$
    else
      Query $x_{n-l}$ followed by $x_l$
    end if
    $l \leftarrow l + 1$
  end while{first phase: alternate branches of tree}
  while evaluation unknown do
    if $p_l \geq p_{n-l} \geq 1 - c$ then
      Query $x_{n-l}$
    else if $c \geq p_l \geq p_{n-l}$ then
      Query $x_l$
    end if
    $l \leftarrow l + 1$
  end while{second phase: single branch in tree}
```

In the Truncated Round Robin, we proceed level by level, in two phases. The first phase concludes once we reach a level l where $p_l > p_{n-l} \geq 1 - c$ or $c \geq p_l > p_{n-l}$. Let ℓ denote this level. In the first phase, we test both x_l and x_{n-l}, testing first the variable whose probability is closest to $\frac{1}{2}$. In the second phase, we abandon the round robin and instead continue down a single branch in the adaptive tree. Specifically, in the second phase, if $p_l > p_{n-l} \geq 1 - c$, then we continue down the right branch, testing the remaining variables in increasing order of p_i. If $c \geq p_l > p_{n-l}$, then we continue down the left branch, testing the remaining variables in decreasing order of p_i. Fixing $c = \frac{3-\sqrt{5}}{2} \approx 0.381966$ in the algorithm, the following holds.

▶ **Theorem 3.** *When all tests have unit cost, the Truncated Round Robin Algorithm achieves an approximation factor of φ for non-adaptive evaluation of the Unanimous Vote function.*

Proof. Consider the optimal adaptive strategy T. It tests a bit x_0 and then follows the optimal AND or OR strategy depending on whether $x_0 = 1$ or $x_0 = 0$. Assume the other bits are indexed so $p_1 \geq p_2 \geq \ldots \geq p_{n-1}$. Thus T is the tree in Figure 1. Let C^*_{adapt} be the expected cost of T. Let $C^*_{non-adapt}$ be the expected cost of the optimal non-adaptive strategy. Let $C_{i,TRR}$ be the cost of running the TRR subroutine in (Algorithm 3) with root x_i. We use x_0 to denote the root of T. Since the TRR algorithm tries all possible roots, its output strategy has expected cost $\min_i C_{i,TRR}$. We will prove the following claim: $C_{0,TRR} \leq \varphi C^*_{adapt}$. Since the expected cost of the optimal adaptive strategy is bounded above by the expected cost of the optimal non-adaptive strategy, the claim implies that $\min_i C_{i,TRR} \leq C_{0,TRR} \leq \varphi C^*_{adapt}$. Further, $C^*_{adapt} \leq C^*_{non-adapt}$, which proves the theorem.

We now prove the claim. We will write the expected cost of the TRR (with root x_0) as $C_{0,TRR} = 1 + E_1 + (1 - P_1)E_2$. Here, E_1 is the expected number of bits tested in T in the first phase (i.e. in levels $l < \ell$), E_2 is the expected number of variables tested among levels in T in the second phase (levels $l \geq \ell$), given that the second phase is reached, and P_1 is the probability of ending during the first phase. Note that the value of ℓ is determined only by the values of the p_i, and it is independent of the test outcomes.

We will write the expected cost of T (the adaptive tree which is optimal w.r.t all trees with root x_0) as $C^*_{adapt} = 1 + E'_1 + (1 - P'_1)E'_2$ where E'_1 is the expected number of bits queried in T before level ℓ, P'_1 is the probability of ending before level ℓ, and E'_2 is the expected number of bits queried in levels ℓ and higher, given that ℓ was reached.

To prove our claim, we will upper bound the ratio $\alpha := \frac{1+E_1+(1-P_1)E_2}{1+E'_1+(1-P'_1)E'_2}$. Recall that since $c < 1/2$, we have $c < 1 - c$. Also, the first phase ends if all bits have been tested, which implies that for all l in the first phase, $l \le \lceil (n-1)/2 \rceil$ so $p_{n-l} \le p_l$. We break the first phase into two parts: (1) The first part consists of all levels l where $p_{n-l} \le c < 1 - c \le p_l$. (2) The second part consists of all levels l where $p_l \in (c, 1 - c)$ or $p_{n-l} \in (c, 1 - c)$, or both.

Let us rewrite the expected cost E_1 as $E_1 = E_{1,1} + (1 - P_{1,1})E_{1,2}$. where $E_{1,1}$ is the expected cost of the first part of phase 1, $E_{1,2}$ is the expected cost of the second part of phase 1, and $P_{1,1}$ is the probability of terminating during the first part of phase 1. Analogously for the cost on tree T, we can rewrite $E'_1 = E'_{1,1} + (1 - P'_{1,1})E'_{1,2}$. Then, the ratio we wish to upper bound becomes $\alpha = \frac{1+E_{1,1}+(1-P_{1,1})E_{1,2}+(1-P_1)E_2}{1+E'_{1,1}+(1-P'_{1,1})E'_{1,2}+(1-P'_1)E'_2}$ which we will upper bound by examining the three ratios

$$\theta_1 := \frac{1 + E_{1,1}}{1 + E'_{1,1}} \qquad \theta_2 := \frac{(1 - P_{1,1})E_{1,2}}{(1 - P'_{1,1})E'_{1,2}} \qquad \theta_3 := \frac{(1 - P_1)E_2}{(1 - P'_1)E'_2}$$

For ratio θ_1, notice that the TRR does at most two tests for every tree level, so $E_{1,1} \le 2E'_{1,1}$, and thus $\frac{1+E_{1,1}}{1+E'_{1,1}} \le \frac{1+2E'_{1,1}}{1+E'_{1,1}}$. Also, $\frac{\mathrm{d}}{\mathrm{d}\,x}\left(\frac{1+2x}{1+x}\right) = \frac{1}{(1+x)^2} > 0$ for $x > 0$. For each path in tree T, for the levels in the first part of the first phase, the probability of getting a result that causes termination is at least $1 - c$. This is because in the first part, $p_l \ge 1 - c > c \ge p_{n-l}$. If we are taking the left branch (because $x_0 = 0$) we terminate when we get a test outcome of 1, and on the right ($x_0 = 1$), we terminate when we get a test outcome of 0. Each bit queried is an independent Bernoulli trial, so $E'_{1,1} \le \frac{1}{1-c}$. Because $\frac{1+2x}{1+x}$ is increasing, we can assert that

$$\theta_1 = \frac{1 + E_{1,1}}{1 + E'_{1,1}} < \frac{1 + 2(1 - c)^{-1}}{1 + 1(1 - c)^{-1}} = \frac{3 - c}{2 - c}.$$

Next we will upper bound the second ratio θ_2. Let $P(l)$ represent the probability of reaching level l in the TRR. Further, let q_l represent the probability of querying the second bit in level l given that we have reached level l. Then, observe that $(1 - P_{1,1})E_{1,2}$ can be written as the sum over all levels l in phase 1, part 2 of $P(l)(1 + q_l)$. Note that in phase 1, the first bit queried is the bit x_i such that p_i is closest to 0.5. Notice also that in the second part of the first phase, each level has at least one variable x_i such that $p_i \in (c, 1 - c)$. This also means that $1 - p_i \in (c, 1 - c)$. This means that the first test performed in any given level in phase 1, part 2 will cause the TRR to terminate with probability at least c. This means that for each level l in this part of the TRR, we will have $q_l \le 1 - c$.

Similarly, $(1 - P'_{1,1})E'_{1,2}$ is the sum over all levels l which comprise phase 1, part 2 in the TRR of $P'(l)$. Here, $P'(l)$ is defined as the probability of reaching level l in tree T. We do not multiply by $1 + q_l$ since in the evaluation of T we only perform one test at each level.

Consider the evaluation of tree T on an assignment. If the evaluation terminates upon reaching level l in the tree, for $l < \ell$, then the evaluation using the TRR must terminate at a level $l' \le l$. That is, the TRR will terminate at level l *or earlier* for the same assignment. Thus, we get that $P(l) \le P'(l)$. Using this, we can achieve the following bound on the second ratio (letting S_2 denote the set of all levels included in the second part of phase 1):

$$\theta_2 = \frac{(1 - P_{1,1})E_{1,2}}{(1 - P'_{1,1})E'_{1,2}} = \frac{\sum_{l\in S_2} P(l)(1 + q_l)}{\sum_{l\in S_2} P'(l)} \le \frac{\sum_{l\in S_2} P(l)(1 + 1 - c)}{\sum_{l\in S_2} P(l)} = 2 - c.$$

Finally, we wish to upper bound the last ratio, $\theta_3 = \frac{(1-P_1)E_2}{(1-P_1')E_2'}$. Let $l^* = \ell$ denote the first level included in the second phase of the TRR. Without loss of generality, assume that $c \geq p_{l^*} \geq p_{n-l^*}$ so that in the TRR, the second phase queries the remaining bits in decreasing order of p_i. Thus, all bits x_i queried in the second phase satisfy $p_i \leq c$. (The argument is symmetric for the case where $p_{l^*} \geq p_{n-l^*} \geq 1-c$).

In this case, any assignments that do not cause termination in the TRR during the first phase, and that have $x_0 = 0$ (i.e., they would go down the left branch of T), will follow the same path through the nodes in left branch, for levels l^* and higher, that they would have followed in the optimal strategy T. (In fact, tests from the right branch of the tree that were previously performed in phase 1 of the TRR do not have to be repeated.)

The numerator of the third ratio θ_3 is equal to the sum, over all assignments x reaching level l^* in the TRR, of $Pr(x)C_2(x)$, where $C_2(x)$ is the total cost of all bits queried in phase 2 for assignment x. Let Q_0 be the subset of assignments reaching level l^* in the TRR which have $x_0 = 0$ and let Q_1 be the subset of assignments reaching level l^* in the TRR which have $x_0 = 1$. Let D_0 represent the sum over all assignments in Q_0 of $Pr(x)C_2(x)$ and let D_1 represent the sum over all assignments in Q_1 of $Pr(x)C_2(x)$. Then, letting S_{l^*} represent the set of assignments reaching level l^* in the TRR, we can rewrite the numerator of the third ratio as $\sum_{x\in S_{l^*}} Pr(x)C_2(x) = \sum_{x\in Q_0} Pr(x)C_2(x) + \sum_{x\in Q_1} Pr(x)C_2(x) = D_0 + D_1$.

The denominator of the third ratio is the sum, over all assignments x reaching level l^* in the tree, of $Pr(x)C_2'(x)$, where $C_2'(x)$ is the total cost of all bits queried in tree T at level l^* and below. Let S_{l^*}' denote the set of assignments x reaching level l^* in tree T. Next, observe that $S_{l^*} \subseteq S_{l^*}'$ since any assignment that reaches level l^* in the TRR must also reach level l^* in the tree. We can again rewrite the denominator as $\sum_{x\in S_{l^*}'} Pr(x)C_2'(x) \geq \sum_{x\in S_{l^*}} Pr(x)C_2'(x) = B_0 + B_1$ where $B_0 = \sum_{x\in Q_0} Pr(x)C_2'(x)$ and $B_1 = \sum_{x\in Q_1} Pr(x)C_2'(x)$. The third ratio θ_3 can thus be upper bounded by $\theta_3 \leq \frac{(1-P_1)E_2}{(1-P_1)E_2} \leq \frac{D_0+D_1}{B_0+B_1}$.

For any $x \in Q_0$, the number of bits queried in level l^* or below in the TRR is less than or equal to the number of bits queried on x in level l^* or below in the tree. Thus $D_0 \leq B_0$.

For $x \in Q_1$, the number of bits queried at level l^* or below is at least one. Thus $B_1 \geq J_1$, where J_1 is the probability that a random assignment x has $x_0 = 1$ and reaches level l^*.

Note that TRR will terminate on an assignment with $x_0 = 1$ when it first tests a bit that has value 0. Also note that each bit x_i in level l^* and below has probability $p_i \leq c$ of having value 1 and thus probability $1 - p_i \geq 1 - c$ of having value 0 and ending the TRR. Since each bit queried is an independent trial, the expected number of bits queried before termination is at most $(1-c)^{-1}$. Thus, $D_1 \leq (1-c)^{-1}J_1$. Together with the fact that $D_0 \leq B_0$, we get $\frac{D_0+D_1}{B_0+B_1} \leq \frac{B_0+(1-c)^{-1}J_1}{B_0+J_1}$. Finally, we observe that since $\frac{B_0}{B_0} = 1$ and $\frac{(1-c)^{-1}J_1}{J_1} \leq \frac{1}{1-c}$, it follows from our earlier upper bound on θ_3, namely $\theta_3 \leq \frac{D_0+D_1}{B_0+B_1}$, that

$$\theta_3 \leq \frac{D_0+D_1}{B_0+B_1} \leq \frac{1}{1-c}.$$

Thus, we have three upper bounds: (1) $\theta_1 \leq \frac{3-c}{2-c}$, (2) $\theta_2 \leq 2-c$, and (3) $\theta_3 \leq \frac{1}{1-c}$. This gives us an upper bound on the ratio of the expected cost of the TRR to the tree T, and thus an upper bound on the approximation factor. This bound is simply the maximum of the three upper bounds: $\frac{1+E_1+(1-P_1)E_2}{1+E_1'+(1-P_1')E_2'} \leq \max\left\{\frac{3-c}{2-c}, 2-c, \frac{1}{1-c}\right\}$. Setting $c = \frac{3-\sqrt{5}}{2} \approx 0.381966$ causes all three upper bounds to equal φ. Thus, running the TRR algorithm with $c = \frac{3-\sqrt{5}}{2}$ produces an expected cost of no more than φ times the expected cost of an optimal strategy. ◀

Table 1 Results for the Adaptive SSClass Problem.

	unit costs	arbitrary costs
weighted	$O(\log W)$-approx [Section 4]; $3(B-1)$ [Section 4]	$O(\log W)$-approx [Section 4]; $3(B-1)$ [Section 4]
unweighted	4-approx [Section 5.3]; $(B-1)$-approx [Section 4]	$O(\log n)$-approx; $(B-1)$-approx [Section 4]
k-of-n function	exact algorithm [known]	exact algorithm [known]
Unanimous Vote function	exact algorithm [Section 5.4]	exact algorithm [Section 5.4]

Table 2 Results for the Non-Adaptive SSClass Problem.

	unit costs	arbitrary costs
weighted	open	open
unweighted	4-approx [Section 5.3]	$2(B-1)$-approx [Section 5.3]
k-of-n function	2-approx [Section 5.3]	2-approx [Section 5.3]
Unanimous Vote function	φ-approx [Section 5.5]	2-approx [Section 5.5]

References

1 Jayadev Acharya, Ashkan Jafarpour, and Alon Orlitsky. Expected query complexity of symmetric Boolean functions. In *IEEE 49th Annual Allerton Conference on Communication, Control, and Computing*, pages 26–29, 2011.

2 Sarah R. Allen, Lisa Hellerstein, Devorah Kletenik, and Tonguç Ünlüyurt. Evaluation of monotone dnf formulas. *Algorithmica*, 77(3):661–685, 2017.

3 Eric Bach, Jérémie Dusart, Lisa Hellerstein, and Devorah Kletenik. Submodular goal value of boolean functions. *Discrete Applied Mathematics*, 238:1–13, 2018. doi:10.1016/j.dam.2017.10.022.

4 Yosi Ben-Dov. Optimal testing procedure for special structures of coherent systems. *Management Science*, 1981.

5 Endre Boros and Tonguç. Ünlüyurt. Diagnosing double regular systems. *Annals of Mathematics and Artificial Intelligence*, 26(1-4):171–191, September 1999. doi:10.1023/A:1018958928835.

6 Ming-Feng Chang, Weiping Shi, and Kent Fuchs, W.˙ Optimal diagnosis procedures for k-out-of-n structures. *IEEE Transactions on Computers*, 39(4):559–564, April 1990.

7 Hirakendu Das, Ashkan Jafarpour, Alon Orlitsky, Shengjun Pan, and Ananda Theertha Suresh. On the query computation and verification of functions. In *IEEE International Symposium on Information Theory (ISIT)*, pages 2711–2715, 2012.

8 Amol Deshpande, Lisa Hellerstein, and Devorah Kletenik. Approximation algorithms for stochastic submodular set cover with applications to boolean function evaluation and min-knapsack. *ACM Trans. Algorithms*, 12(3):42:1–42:28, April 2016. doi:10.1145/2876506.

9 Dimitrios Gkenosis, Nathaniel Grammel, Lisa Hellerstein, and Devorah Kletenik. The stochastic score classification problem. *CoRR*, 2018. arXiv:1806.10660.

10 Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486, 2011.

11 Daniel Golovin and Andreas Krause. Adaptive submodularity: A new approach to active learning and stochastic optimization (version 5). *CoRR*, abs/1003.3967, 2017. arXiv:1003.3967.

12 Nathaniel Grammel, Lisa Hellerstein, Devorah Kletenik, and Patrick Lin. Scenario submodular cover. In *Proceedings of the 14th International Workshop on Approximation and Online Algorithms*, pages 116–128. Springer, 2016.

13 Russell Greiner, Ryan Hayward, Magdalena Jankowska, and Michael Molloy. Finding optimal satisficing strategies for and-or trees. *Artificial Intelligence*, 170(1):19–58, 2006.

14 Anupam Gupta and Viswanath Nagarajan. A stochastic probing problem with applications. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 205–216. Springer, 2013.

15 Jongbin Jung, Connor Concannon, Ravi Shroff, Sharad Goel, and Daniel G Goldstein. Simple rules for complex decisions. *arXiv preprint arXiv:1702.04690*, 2017.

16 Prabhanjan Kambadur, Viswanath Nagarajan, and Fatemeh Navidi. Adaptive submodular ranking. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 317–329. Springer, 2017.

17 Hemant Kowshik and PR Kumar. Optimal computation of symmetric boolean functions in collocated networks. *IEEE Journal on Selected Areas in Communications*, 31(4):639–654, 2013.

18 Feng Nan and Venkatesh Saligrama. Comments on the proof of adaptive stochastic set cover based on adaptive submodularity and its implications for the group identification problem in "group-based active query selection for rapid diagnosis in time-critical situations". *IEEE Trans. Information Theory*, 63(11):7612–7614, 2017. `doi:10.1109/TIT.2017.2749505`.

19 Salam Salloum. *Optimal testing algorithms for symmetric coherent systems.* PhD thesis, University of Southern California, 1979.

20 Salam Salloum and Melvin Breuer. An optimum testing algorithm for some symmetric coherent systems. *Journal of Mathematical Analysis and Applications*, 101(1):170 – 194, 1984. `doi:10.1016/0022-247X(84)90064-7`.

21 Salam Salloum and Melvin A. Breuer. Fast optimal diagnosis procedures for k-out-of-n:g systems. *IEEE Transactions on Reliability*, 46(2):283–290, Jun 1997. `doi:10.1109/24.589958`.

22 Sahil Singla. The price of information in combinatorial optimization. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2523–2532. SIAM, 2018.

23 Truyen Tran, Wei Luo, Dinh Phung, Jonathan Morris, Kristen Rickard, and Svetha Venkatesh. Preterm birth prediction: Deriving stable and interpretable rules from high dimensional data. In *Conference on Machine Learning in Healthcare, LA, USA*, 2016.

24 Tonguç Ünlüyurt. Sequential testing of complex systems: a review. *Discrete Applied Mathematics*, 142(1-3):189–205, 2004.

25 Berk Ustun and Cynthia Rudin. Supersparse linear integer models for optimized medical scoring systems. *Machine Learning*, 102(3):349–391, 2016.

26 Berk Ustun and Cynthia Rudin. Optimized risk scores. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1125–1134. ACM, 2017.

27 Jiaming Zeng, Berk Ustun, and Cynthia Rudin. Interpretable classification models for recidivism prediction. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 180(3):689–722, 2017.

Improved Space-Time Tradeoffs for kSUM

Isaac Goldstein[1]
Bar-Ilan University, Ramat Gan, Israel
goldshi@cs.biu.ac.il

Moshe Lewenstein[2]
Bar-Ilan University, Ramat Gan, Israel
moshe@cs.biu.ac.il

Ely Porat[3]
Bar-Ilan University, Ramat Gan, Israel
poratelу@cs.biu.ac.il

Abstract

In the kSUM problem we are given an array of numbers $a_1, a_2, ..., a_n$ and we are required to determine if there are k different elements in this array such that their sum is 0. This problem is a parameterized version of the well-studied SUBSET-SUM problem, and a special case is the 3SUM problem that is extensively used for proving conditional hardness. Several works investigated the interplay between time and space in the context of SUBSET-SUM. Recently, improved time-space tradeoffs were proven for kSUM using both randomized and deterministic algorithms.

In this paper we obtain an improvement over the best known results for the time-space tradeoff for kSUM. A major ingredient in achieving these results is a general self-reduction from kSUM to mSUM where $m < k$, and several useful observations that enable this reduction and its implications. The main results we prove in this paper include the following: (i) The best known Las Vegas solution to kSUM running in approximately $O(n^{k-\delta\sqrt{2k}})$ time and using $O(n^\delta)$ space, for $0 \le \delta \le 1$. (ii) The best known deterministic solution to kSUM running in approximately $O(n^{k-\delta\sqrt{k}})$ time and using $O(n^\delta)$ space, for $0 \le \delta \le 1$. (iii) A space-time tradeoff for solving kSUM using $O(n^\delta)$ space, for $\delta > 1$. (iv) An algorithm for 6SUM running in $O(n^4)$ time using just $O(n^{2/3})$ space. (v) A solution to 3SUM on random input using $O(n^2)$ time and $O(n^{1/3})$ space, under the assumption of a random read-only access to random bits.

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

Keywords and phrases kSUM, space-time tradeoff, self-reduction

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.37

Related Version A full version of the paper is available at `https://arxiv.org/abs/1807.03718`.

1 Introduction

In the kSUM problem we are given an array of numbers $a_1, a_2, ..., a_n$ and we are required to determine if there are k different elements in this array such that their sum equals 0. This is a parameterized version of SUBSET-SUM, one of the first well-studied NP-complete problems,

[1] This research is supported by the Adams Foundation of the Israel Academy of Sciences and Humanities.
[2] This work was partially supported by an ISF grant #1278/16, BSF grant 2010437 and a GIF grant 1147/2011.
[3] This work was partially supported by an ISF grant #1278/16.

© Isaac Goldstein, Moshe Lewenstein, and Ely Porat;
licensed under Creative Commons License CC-BY

which also can be thought of as a special case of the famous KNAPSACK problem [18]. A special case of kSUM is the 3SUM problem which is extensively used to prove conditional lower bounds for many problems, including: string problems [3, 5, 14, 19], dynamic problems [2, 21], computational geometry problems [9, 13], graph problems [1, 4, 19] etc.

The kSUM problem can be trivially solved in $O(n^k)$ time using $\tilde{O}(1)$ space (for constant k), or in $O(n^{\lceil k/2 \rceil})$ time using $O(n^{\lceil k/2 \rceil})$ space. It is known that there is no solution to kSUM with $n^{o(k)}$ running time, unless the Exponential Time Hypothesis is false [22]. However, a central goal is to find the best tradeoff between time and space for kSUM. Specifically, it is interesting to have a full understanding of questions like: What is the best running time we can achieve by allowing at most *linear* space? How can the running time be improved by using $O(n^2)$, $O(n^3)$ or $O(n^{10})$ space? Can we get any improvement over $O(n^k)$ running time for almost constant space or use less space for $O(n^{\lceil k/2 \rceil})$ time solution? What is the best time-space tradeoff for interesting special cases like 3SUM? Questions of this type guided a line of research work and motivate our paper.

One of the first works on the time-space tradeoff of kSUM and SUBSET-SUM is by Shamir and Schroeppel [23]. They showed a simple reduction from SUBSET-SUM to kSUM. Moreover, they presented a deterministic solution to 4SUM running in $O(n^2)$ time using $O(n)$ space. They used these solution and reduction to present an $O^*(2^{n/2})$ time and $O^*(2^{n/4})$ space algorithm for SUBSET-SUM. Furthermore, they demonstrate a space-time tradeoff curve for SUBSET-SUM by a generalized algorithm. More recently, a line of research work improved the space-time tradeoff of Shamir and Schroeppel by using randomization. This includes works by Howgrave-Graham and Joux [17], Becker et al. [10] and Dinur et al. [12] on random instances of SUBSET-SUM, and a matching tradeoff curve for worst-case instances of SUBSET-SUM by Austrin et al. [6].

Wang [24] used randomized techniques to improve the space-time tradeoff curve for kSUM. Specifically, he presented a Las Vegas randomized algorithm for 3SUM running in $\tilde{O}(n^2)$ time using just $\tilde{O}(\sqrt{n})$ space. Moreover, for general k he demonstrated a Monte Carlo algorithm for kSUM that uses $O(n^\delta)$ space using approximately $O(n^{k-\delta\sqrt{2k}})$ time, for $0 \leq \delta \leq 1$. Lincoln et al. [20] achieved $O(n^2)$ time and $\tilde{O}(\sqrt{n})$ space *deterministic* solution for 3SUM. For general kSUM ($k \geq 4$), they obtained a deterministic algorithm running in $O(n^{k-3+4/(k-3)})$ time using linear space and $O(n^{k-2+2/k})$ time using $O(\sqrt{n})$ space.

Very recently, Bansal et al. [7] presented a randomized solution to SUBSET-SUM running in $O^*(2^{0.86n})$ time and using just polynomial space, under the assumption of a random read-only access to exponentially many random bits. This is based on an algorithm that determines whether two given lists of length n with integers bounded by a polynomial in n share a common value. This problem is closely related to 2SUM and they proved it can be solved using $O(\log n)$ space in significantly less than $O(n^2)$ time if no value occurs too often in the same list (under the assumption of a random read-only access to random bits). They also used this algorithm to obtain an improved solution for kSUM on random input.

Finally, it is worth mentioning that recent works by Goldstein et al. [15, 16] consider the space-time tradeoff of data structures variants of 3SUM and other related problems.

1.1 Our Results

In this paper we improve the best known bounds for solving kSUM in both (Las Vegas) randomized and deterministic settings. A central component in our results is a general *self-reduction* from kSUM to mSUM for $m < k$:

▶ **Theorem 1.** *There is a self-reduction from one instance of kSUM with n integers in each array to $O(n^{(k/m-1)(m-\delta)})$ instances of mSUM (reporting) with n integers in each array and $O(n^{(k/m-1)(m-\delta)})$ instances of $\lceil \frac{k}{m} \rceil$SUM with n^{δ} integers in each array, for any integer $m < k$ and $0 < \delta \leq m$.*

Moreover, we present several crucial observations and techniques that play central role in this reduction and other results of this paper.

For general kSUM we obtain the following results:

Using our self-reduction scheme and the ideas by Lincoln et al. [20], we obtain a deterministic solution to kSUM that significantly improves over the deterministic algorithm by Lincoln at al. [20] that runs in $O(n^{k-3+4/(k-3)})$ time using linear space and $O(n^{k-2+2/k})$ time using $O(\sqrt{n})$ space:

▶ **Theorem 2.** *For $k \geq 2$, kSUM can be solved by a deterministic algorithm that runs in $O(n^{k-\delta g(k)})$ time using $O(n^{\delta})$ space, for $0 \leq \delta \leq 1$ and $g(k) \geq \sqrt{k} - 2$.*

By allowing randomization we have the following result:

▶ **Theorem 3.** *For $k \geq 2$, kSUM can be solved by a Las Vegas randomized algorithm that runs in $O(n^{k-\delta f(k)})$ time using $O(n^{\delta})$ space, for $0 \leq \delta \leq 1$ and $f(k) \geq \sqrt{2k} - 2$.*

Our Las Vegas algorithm has the same running time and space as Wang's [24] Monte Carlo algorithm. The idea is to modify his algorithm using the observations and techniques from our self-reduction scheme.

We also consider solving kSUM using $O(n^{\delta})$ space for $\delta > 1$. Using our self-reduction technique and the algorithm from Theorem 3, we prove the following:

▶ **Theorem 4.** *For $k \geq 2$, kSUM can be solved by a Las Vegas algorithm that runs in $O(n^{k-\sqrt{\delta} f(k)})$ time using $O(n^{\delta})$ space, for $\frac{k}{4} \geq \delta > 1$ and $f(k) \geq \sqrt{2k} - 2$.*

Our self-reduction technique can also be applied directly to obtain improvements on the space-time tradeoff for special cases of kSUM. Especially interesting is the case of 6SUM which can be viewed as a combination of the "easy" 4SUM and the "hard" 3SUM. We obtain randomized algorithms solving 6SUM in $O(n^3)$ time using $O(n^2)$ space and in $O(n^4)$ time using just $O(n^{2/3})$ space (and not $O(n)$ as known by previous methods [24]).

Finally, combining our techniques with the techniques by Bansal et al. [7] we obtain improved space-time tradeoffs for some special cases of kSUM on random input, under the assumption of a random read-only access to random bits. One notable result of this flavour is a solution to 3SUM on random input that runs in $O(n^2)$ time and $O(n^{1/3})$ space, instead of the $O(n^{1/2})$ space solutions known so far [20, 24]. The last results regarding kSUM on random input appear in the full version of this paper.

2 Preliminaries

In the basic definition of kSUM the input contains just one array. However, in a variant of this problem, which is commonly used, we are given k arrays of n numbers and we are required to determine if there are k elements, one from each array, such that their sum equals 0. It is easy to verify that this variant is equivalent to kSUM in terms of time and space complexity. We also note that the choice of 0 is not significant, as it can be easily shown that the problem is equivalent in terms of time and space complexity even if we put any other constant t, called the *target number*, instead of 0. Throughout this paper we consider kSUM with k arrays and a target value t. We also consider the *reporting* version of kSUM in which we need to report *all* subsets of k elements that sum up to 0 or some other constant t.

All the randomized algorithms in this paper solve kSUM on input arrays that contain *integer* numbers. The target number t is also assumed to be an integer. This assumption was also used in previous papers considering the space-time tradeoff for kSUM (see [24]). The deterministic solution we present is the only one that works even for kSUM on real numbers.

Let $\mathcal{H}$ be a family of hash functions from $[u]$ to $[m]$ ($[u]$ is some unbounded universe). $\mathcal{H}$ is called *linear* if for any $h \in \mathcal{H}$ and any $x_1, x_2 \in [u]$, we have $h(x_1)+h(x_2) \equiv h(x_1+x_2) \pmod m$. $\mathcal{H}$ is called *almost-linear* if for any $h \in \mathcal{H}$ and any $x_1, x_2 \in [u]$, we have either $h(x_1)+h(x_2) \equiv h(x_1+x_2)+c_h \pmod m$, or $h(x_1)+h(x_2) \equiv h(x_1+x_2)+c_h+1 \pmod m$, where c_h is an integer that depends only on the choice of h. Throughout this paper we will assume that h is linear as almost linearity will just add a constant factor cost to the running time and a change in the offsets which can be easily handled. For a function $h : [u] \to [m]$ and a set $S \subset [u]$ where $|S| = n$, we say that $i \in [m]$ is an overflowed value of h if $|\{x \in S : h(x) = i\}| > 3n/m$. $\mathcal{H}$ is called *almost-balanced* if for a random $h \in \mathcal{H}$ and any set $S \subset [u]$ where $|S| = n$, the expected number of elements from S that are mapped to overflowed values is $O(m)$ (for more details see [8, 11, 19, 24]). There are concrete constructions of hash families that are almost-linear and almost-balanced [19, 24]. In the Las Vegas algorithms in this paper, we assume, in order for the presentation to be clear, that an almost-balanced hash function can become balanced (which means that there are no overflowed values at all). The full details of how this can be done in our Las Vegas algorithm appear in the full version of this paper.

3 Self-Reduction From kSUM to mSUM

We demonstrate a general efficient reduction from a single instance of kSUM to many instances of mSUM (reporting) and $\lceil \frac{k}{m} \rceil$SUM for $m < k$:

▶ **Theorem 1.** *There is a self-reduction from one instance of kSUM with n integers in each array to $O(n^{(k/m-1)(m-\delta)})$ instances of mSUM (reporting) with n integers in each array and $O(n^{(k/m-1)(m-\delta)})$ instances of $\lceil \frac{k}{m} \rceil$SUM with $O(n^{\delta})$ integers in each array, for any integer $m < k$ and $0 < \delta \leq m$.*

Proof. Given an instance of kSUM that contains k arrays $A_1, A_2, ..., A_k$ with n integers in each of them and a target number t, we do the following (for now, we assume that k is a multiple of m. Notice that k and m are considered as constants):

1. Partition the k arrays into k/m groups of m arrays. We denote the ith group in this partition by G_i.
2. Pick an almost-linear almost-balanced hash function $h : [u] \to [n^{m-\delta}]$ and apply it to each element in every array ($[u]$ is some unbounded universe).
3. For each possible choice of $t_1, t_2, ..., t_{k/m-1} \in [n^{m-\delta}]$:
   - **3.1** Find in each group G_i all m-tuples $(a^{j_1}_{(i-1)m+1}, a^{j_2}_{(i-1)m+2}, ..., a^{j_m}_{im})$, where a^j_x is the jth element in A_x, such that $h(a^{j_1}_{(i-1)m+1} + a^{j_2}_{(i-1)m+2} + ... + a^{j_m}_{im}) = t_i$. We can find these m-tuples by solving mSUM reporting with group G_i (after applying h) and the target number t_i. All m-tuples that are found are saved in a list L_i (L_i contains m-tuples that are found for a specific choice of t_i, after this choice is checked, as explained below, they are replaced by m-tuples that are found for a new choice of t_i).
   - **3.2** For $G_{k/m}$, find all m-tuples $(a^{j_1}_{k-m+1}, a^{j_2}_{k-m+2}, ..., a^{j_m}_{k})$, such that $h(a^{j_1}_{k-m+1} + a^{j_2}_{k-m+2} + ... + a^{j_m}_{k}) = t_{k/m}$. We can find these m-tuples by solving mSUM reporting with group $G_{k/m}$ (after applying h) and the target number $t_{k/m}$.

All m-tuples that are found are saved in the list $L_{k/m}$. The value of the target number $t_{k/m}$ is fully determined by the values of t_i we choose for the other groups, as the overall sum must be $h(t)$ in order for the original sum of elements to be t. Therefore, for $G_{k/m}$ the target value is $t_{k/m} = h(t) - \sum_{i=1}^{k/m-1} t_i$.

3.3 For every $i \in [k/m]$, create an array B_i. For each m-tuple in L_i, add the sum of the elements of this tuple to B_i.

3.4 Solve a $\frac{k}{m}$SUM instance with arrays $B_1, B_2, ..., B_{k/m}$ and the target value t. If there is a solution to this $\frac{k}{m}$SUM instance return 1 - there is a solution to the original kSUM instance.

4. Return 0 - there is no solution to the original kSUM instance.

Correctness. If the original kSUM instance has a solution $a_1 + a_2 + ... + a_k = t$ such that $a_i \in A_i$ for all $i \in [k]$, then this solution can be partitioned to k/m sums: $a_1+a_2+...+a_m = t'_1$, $a_{m+1} + a_{m+2} + ... + a_{2m} = t'_2$,..., $a_{k-m+1} + a_{k-m+2} + ... + a_k = t'_{k/m}$ for some integers $t'_1, t'_2, ...t'_{k/m}$ such that $t'_{k/m} = t - \sum_{i=1}^{k/m-1} t'_i$. Therefore, by applying a hash function h, there is a solution to the original kSUM instance only if there are $t_1, t_2, ..., t_{k/m-1} \in [n^{m-\delta}]$ such that: (a) $h(a_1+a_2+...+a_m) = t_1$, $h(a_{m+1}+a_{m+2}+...+a_{2m}) = t_2$,..., $h(a_{k-m+1}+a_{k-m+2}+...+a_k) = t_{k/m}$ (b) For all i, $t_i = h(t'_i)$. This is exactly what is checked in step 3. However, as the hash function h may cause false-positives (that is, we may have $t_1, t_2, ..., t_{k/m-1} \in [n^{m-\delta}]$ such that their sum is $h(t)$ and $h(a_1+a_2+...+a_m) = t_1$, $h(a_{m+1}+a_{m+2}+...+a_{2m}) = t_2$,..., $h(a_{k-m+1} + a_{k-m+2} + ... + a_k) = t_{k/m}$, but $a_1 + a_2 + ... + a_k \neq t$), we need to verify each candidate solution. This is done in step (3.4).

The correctness of using mSUM (reporting) in steps (3.1) and (3.2) is due to the *linearity* property of h (see the note in Section 2). This linearity implies that finding all m-tuples in G_i such that $h(a^{j_1}_{(i-1)m+1} + a^{j_2}_{(i-1)m+2} + ... + a^{j_m}_{im}) = t_i$ is equivalent to finding all m-tuples in G_i such that $h(a^{j_1}_{(i-1)m+1}) + h(a^{j_2}_{(i-1)m+2}) + ... + h(a^{j_m}_{im}) = t_i$.

Regarding steps (3.3) and (3.4) we have the following observation:

▶ **Observation 1.** *The number of m-tuples that are saved in steps (3.1) and (3.2) in some L_i for each possible value of t_i is no more than $O(n^\delta)$.*

The total number of m-tuples in some group G_i is n^m. As h is an almost-*balanced* hash function (that can become balanced as it is explained in detail in the full version of this paper) with range $[n^{m-\delta}]$, the number of m-tuples that h applied to the sum of their elements equals t_i is expected to be at most $O(n^\delta)$. However, this is true only if all these m-tuples have a different sum of elements. Unfortunately, there may be many m-tuples that the sum of their elements is equal, so all these m-tuples are mapped by h to the same value t_i. Nevertheless, tuples with equal sum of elements are all the same for our purposes (we do not need duplicate elements in any B_i), as we are interested in the *sum* of elements from all arrays no matter which specific elements sum up to it.

That being said, in steps (3.1) and (3.2) we do not add to L_i every m-tuple that the sum of the elements of this tuple is t_i. Instead, for each m-tuple that h over the sum of its elements equals t_i, we check if there is already a tuple with the same sum of elements in L_i and only if there is no such tuple we add our m-tuple to L_i. In order to efficiently check for the existence of an m-tuple with the same sum in L_i, we can save the elements of L_i in a balanced search tree or use some dynamic perfect hashing scheme. We call the process of removing m-tuples with same sum from L_i the **removing duplicate sums** process.

The total number of mSUM and $\frac{k}{m}$SUM instances is determined by the number of possible choices for $t_1, t_2, ..., t_{k/m-1}$ that is $O(n^{(k/m-1)(m-\delta)})$. Notice that k and m are fixed constants.

Modifications in the self-reduction for k that is not a multiple of m. In case k is not a multiple of m, we partition the k arrays into $\lceil k/m \rceil$ groups such that some of them have m arrays and the others have $m-1$ arrays. In any case when we partition into groups of unequal size the range of the hash function h is determined by the smallest group. If the smallest group has d arrays then we use $h : [u] \to [n^{d-\delta}]$. Using this h for groups of size d, we get all d-tuples that h applied to their sum of elements equals some constant t_i. We expect $O(n^\delta)$ such tuples (if we exclude d-tuples with the same sum as explained previously). However, for groups with more than d arrays, say $d+\ell$, we expect the number of $(d+\ell)$-tuples that h applied to their sum of elements equals t_i to be $O(n^{\ell+\delta})$. Therefore, in order to just save all these tuples we must spend more space than we can afford to use. Therefore, we will only save $O(n^\delta)$ of them in each time.

However, in order to be more efficient, we do not start solving $(d+\ell)$SUM reporting for every $O(n^\delta)$ tuples we report on. Instead, we solve $(d+\ell)$SUM reporting once for all the expected $O(n^{\ell+\delta})$ $(d+\ell)$-tuples that h applied to their sum of elements equals t_i. We do so by pausing the execution of $(d+\ell)$SUM reporting whenever we report on $O(n^\delta)$ tuples. After handling the reported tuples we resume the execution of the paused $(d+\ell)$SUM reporting. We call this procedure of reporting on demand a partial output of the recursive calls, the **paused reporting** process.

As noted before, the number of $(d+\ell)$-tuples that h applied to their sum of elements equals t_i may be greater than $O(n^{\ell+\delta})$, because there can be many $(d+\ell)$-tuples that the sum of their elements is equal. We argued that we can handle this by saving only those tuples that the sum of their elements is unequal. However, in our case we save only $O(n^\delta)$ tuples out of $O(n^{\ell+\delta})$ tuples, so we do not have enough space to make sure we do not save tuples that their sums were already handled. Nevertheless, the fact that we repeat handling tuples with the same sum of elements is not important since we anyway go over all possible tuples in our $(d+\ell)$SUM instance. The only crucial point is that in the last group that its target number is fixed, we have only $O(n^\delta)$ elements for each $t_{\lceil k/m \rceil}$. This is indeed what happens if we take that group to be the group with the d arrays (the smallest group). We call this important observation the **small space of fixed group** observation. That being said, our method can be applied even in case we partition to groups of unequal number of arrays. ◀

Using this self-reduction scheme we obtain the following Las Vegas solution to kSUM:

▶ **Lemma 5.** *For $k \geq 2$, kSUM can be solved by a Las Vegas algorithm following a self-reduction scheme that runs in $O(n^{k-\delta f(k)})$ time using $O(n^\delta)$ space, for $0 \leq \delta \leq 1$ and $f(k) \geq \sum_{i=1}^{\log\log k} k^{1/2^i} - \log\log k - 2$.*

Proof. Using our self-reduction from Theorem 1, we can reduce a single instance of kSUM to many instances of mSUM and $\frac{k}{m}$SUM for $m < k$. These instances can be solved recursively by applying the reduction many times.

Solving the base case. The base case of the recursion is 2SUM that can be solved in the following way: Given two arrays A_1 and A_2, each containing n numbers, our goal is to find all pairs of elements (a_1, a_2) such that $a_1 \in A_1, a_2 \in A_2$ and $a_1 + a_2 = t$. This can be done easily in $\tilde{O}(n)$ time and $O(n)$ space by sorting A_2 and finding for each element in A_1 a matching element in A_2 using binary search. If we want to use only $O(n^{\delta})$ space we can do it by sorting only $O(n^{\delta})$ elements from A_2 each time and finding among the elements of A_1 a matching pair. This is done by scanning all elements of A_1 and binary searching the sorted portion of A_2. The total time for this procedure is $\tilde{O}(n^{2-\delta})$. Using hashing, following the general scheme we described previously, we can also obtain the same space-time tradeoff. We apply $h : [u] \rightarrow [n^{1-\delta}]$ to all elements of A_1 and A_2. For each value $t_1 \in [n^{1-\delta}]$ we find all elements $a_i \in A_1$ such that $h(a_i) = t_1$ and all elements $a_i \in A_2$ such that $h(a_i) = h(t) - t_1$. These elements form two arrays with $O(n^{\delta})$ elements in expectation, as we use an almost balanced hash function. These arrays serve as a 2SUM instance with $O(n^{\delta})$ elements, which we can solve by the regular (almost) linear time and linear space algorithm mentioned before. That being said, we get an $\tilde{O}(n^{2-\delta})$ time and $O(n^{\delta})$ space algorithm to solve 2SUM for any $0 \leq \delta \leq 1$.

We now analyse the running time of this solution. Denote by $T(k, n, s)$ the time needed to solve kSUM on input arrays of size n with space usage at most $O(s)$. The full recursive process we have described to solve kSUM uses $O(n^{\delta})$ space (notice that the number of levels in the recursion depends only on k that is considered constant) with the following running time: $T(k, n, n^{\delta}) = n^{(k/m-1)(m-\delta)}(T(m, n, n^{\delta}) + T(k/m, n^{\delta}, n^{\delta}))$. In order to solve the running time recursion, we start by solving it for the case that $\delta = 1$. For this case we have that $T(k, n, n) = n^{(k/m-1)(m-1)}(T(m, n, n) + T(k/m, n, n))$. The best running time in this case is obtained by balancing the two expressions within the parenthesis, which is done by setting $m = \sqrt{k}$. We have that $T(k, n, n) = 2n^{(\sqrt{k}-1)(\sqrt{k}-1)}T(\sqrt{k}, n, n) = 2n^{k-2\sqrt{k}+1}T(\sqrt{k}, n, n)$. Solving this recursion we get that $T(k, n, n) = O(n^{k-\sum_{i=1}^{\log\log k} k^{1/2^i}+\log\log k})$.

Now, that we have solved the linear space case we can obtain a solution for any $\delta < 1$ by plugging in this last result in our recursion. We have that $T(k, n, n^{\delta}) =$ $n^{(k/m-1)(m-\delta)}(T(m, n, n^{\delta}) + T(k/m, n^{\delta}, n^{\delta})) =$ $n^{(k/m-1)(m-\delta)}(T(m, n, n^{\delta}) + O(n^{\delta(k-\sum_{i=1}^{\log\log k} k^{1/2^i}+\log\log k)}))$. It turns out that the best running time is obtained by setting $m = 1$. For this value of m we have that $T(k, n, n^{\delta}) =$ $n^{(k-1)(1-\delta)}(T(1, n, n^{\delta}) + O(n^{\delta(k-\sum_{i=1}^{\log\log k} k^{1/2^i}+\log\log k)})) =$ $n^{(k-1)(1-\delta)}(O(n) + O(n^{\delta(k-\sum_{i=1}^{\log\log k} k^{1/2^i}+\log\log k)})) = O(n^{k-\delta\sum_{i=1}^{\log\log k} k^{1/2^i}+\delta(\log\log k+1)-1})$. ◀

Our self-reduction for the case $m = 1$ becomes identical to the one presented by Wang [24]. However, the reduction by Wang is a reduction from kSUM to kSUM on a smaller input size, whereas our reduction is a general reduction from kSUM to mSUM for any $m < k$. Therefore, Wang has to present a different algorithm (discussed later in this paper) to solve kSUM using linear space. However, as this algorithm is Monte Carlo the whole solution is Monte Carlo. Using our generalized self-reduction we obtain a complete solution to kSUM. We have a linear space solution by choosing $m = \sqrt{k}$ and then we can use it to obtain a Las Vegas solution to kSUM for any $\delta \leq 1$ by choosing $m = 1$.

Regarding the self-reduction and its implications we should emphasize three points. The first one concerns our removing duplicate sums process. We emphasize that each time we remove a duplicate sum we regard to the *original* values of the elements within that sum. An important point to observe is that duplicate sums that are caused by any hash function along the recursion, which are not duplicate sums according to the original values, do not affect the running time of our reduction. This is because the range of a hash function in a higher level of the recursion is larger than the total number of tuples we have in lower levels of the recursion. Thus, the number of duplicate sums that are caused by some hash function along the recursion is not expected to be more than $O(1)$. The second issue that we point out is the reporting version of kSUM and the output size. In our reduction in the top level of the recursion we solve kSUM without the need to report on all solutions. In all other levels of the recursion we have to report on all solutions (expect for duplicate sums). In our analysis we usually omit all references to the output size in the running time (and interchange between kSUM and its reporting variant). This is because the total running time that is required in order to report on all solutions is no more than $O(n^m)$ (for all levels of recursion), which does not affect the total running time as $m \leq k/2$. The third issue concerns rounding issues. In the proof of the general self-reduction we presented a general technique of how to handle the situation where k is not a multiple of m. In order to make presentation clear we omit any further reference to this issue in the proof of the last lemma and the theorem in the next section. However, we emphasize that in the worst case the rounding issue may cause an increase by one in the exponent of the running time of the linear space algorithm. This is justified by the fact that the running time of the linear space algorithm is increased by one in the exponent or remains the same as we move from solving kSUM to solving $(k+1)$SUM. Moreover, the gap between two values of k, that the exponent of the running time does not change as we move from solving kSUM to $(k+1)$SUM, increases as a function of k. With that in mind, we decrease by one the lower bound on $f(k)$ and $g(k)$ in last lemma and the next theorem.

In the following sections we present other benefits of our general self-reduction scheme.

4 Improved Deterministic Solution for kSUM

Using the techniques of [20] our randomized solution can be transformed to a deterministic one by imitating the hash function behaviour in a deterministic way. This way we get the following result:

▶ **Theorem 2.** *For $k \geq 2$, kSUM can be solved by a deterministic algorithm that runs in $O(n^{k-\delta g(k)})$ using $O(n^\delta)$ space, for $0 \leq \delta \leq 1$ and $g(k) \geq \sqrt{k} - 2$.*

Proof. We partition the k arrays into k/m groups of m arrays. We denote the ith group in this partition by G_i. For every group G_i, there are n^m sums of m elements, such that each element is from a different array of the m arrays in G_i. We denote by $SUMS_{G_i}$ the array that contains all these sums. A *sorted part* of a group G_i is a continuous portion of the sorted version of $SUMS_{G_i}$. The main idea for imitating the hash function behaviour in a deterministic way is to focus on sorted parts of size n^δ, one for each of the first $k/m - 1$ groups. Then the elements from the last group that are candidates to complete the sum to the target number are fully determined. Each time we pick different n^δ elements out of these elements and form an instance of $(\frac{k}{m})$SUM such that the size of each array is n^δ. The crucial point is that the total number of these instances will be $O(n^{(k/m-1)(m-\delta)})$ as in the solution that uses hashing techniques. This is proven based on the domination lemma of [20] (see the full details in Section 3.1 of [20]). Lincoln et al. [20] present a corollary of the domination lemma as follows: *Given a kSUM instance L, suppose L is divided into g groups $L_1, ..., L_g$ where $|L_i| = n/g$ for all i, and for all $a \in L_i$ and $b \in L_{i+1}$ we have $a \leq b$. Then there are $O(k \cdot g^{k-1})$ subproblems L' of L such that the smallest kSUM of L' is less than zero and the largest kSUM of L' is greater than zero.* Following our scheme, g in this corollary equals $n^{m-\delta}$ in our case (there are g groups of size n^δ in each $SUMS_{G_i}$) and the k in the corollary is in fact k/m in our case. Therefore, we get that the total number of instances that have to be checked is indeed $O(n^{(k/m-1)(m-\delta)})$.

In order for this idea to work, we need to obtain a sorted part of size n^δ from each group G_i. In this case, we do not have the recursive structure as in the randomized solution because we no longer seek for m elements in each group that sum up to some target number, but rather we would like to get a sorted part of each group. However, we can still gain from the fact that we have only $O(n^{(k/m-1)(m-\delta)})$ instances of $(\frac{k}{m})$SUM.

Lincoln et al. [20] presented a simple data structure that obtains a sorted part of size $O(S)$ from an array with n elements using $O(n)$ time and $O(S)$ space. We can use this data structure in order to obtain a sorted part of n^δ elements for each group G_i by considering the elements of the array $SUMS_{G_i}$. Consequently, a sorted part of n^δ elements from G_i can be obtained using $O(n^m)$ time and $O(n^\delta)$ space.

Putting all parts together we have a deterministic algorithm that solves kSUM with the following running time: $T(k,n,n^\delta) = n^{(k/m-1)(m-\delta)}(n^m + T(k/m, n^\delta, n^\delta))$. By setting $m = \sqrt{k}$ we have $T(k,n,n^\delta) = n^{k-\sqrt{k}-\sqrt{k}\delta+\delta}(n^{\sqrt{k}} + T(\sqrt{k}, n^\delta, n^\delta))$. Solving kSUM using linear space can be trivially done using n^k time. Therefore, we get that $T(k,n,n^\delta) = n^{k-\sqrt{k}-\sqrt{k}\delta+\delta}(n^{\sqrt{k}} + n^{\delta\sqrt{k}}) = n^{k-\sqrt{k}\delta+\delta}$. ◀

The last theorem is a significant improvement over the previous results of Lincoln et al. [20] that obtain just a small improvement of at most 3 in the exponent over the trivial solution that uses n^k time, whereas our solution obtains an improvement of almost $\sqrt{k}\delta$ in the exponent over the trivial solution.

5 Las Vegas Variant of Wang's Linear Space Algorithm

Wang [24] presented a Monte Carlo algorithm that solves (T_j+1)SUM in $O(n^{T_{j-1}+1})$ time and linear space, where $T_j = \sum_{i=1}^{j} i$. We briefly sketch his solution here in order to explain how to modify it in order to obtain a Las Vegas algorithm instead of a Monte Carlo algorithm. Given an instance of kSUM with k arrays $A_1, A_2, ..., A_k$ such that $k = T_j + 1$, he partitions the arrays into two groups. The left group contains the first j arrays and the right group all the other arrays. An almost-linear almost-balanced hash function h is chosen, such that its range is $m' = \Theta(n^{j-1})$. The hash function h is applied to all elements in all input arrays. Then, the algorithm goes over all possible values $v_l \in [m']$. For each such value, the first array of the left group is sorted and for all possible sums of elements from the other $j-1$ arrays (one element from each array) it is checked (using binary search) if there is an element from the first array that completes this sum to v_l. If there are j elements that their hashed values sum up to v_l they (the original values) are saved in a lookup table T. At most $\Theta(n)$ entries are stored in T. After handling the left group the right group is handled. Specifically, if the target value is t the sum of elements from the arrays in the right group should be $h(t) - v_l$ (to be more accurate as our hash function is almost linear we have to check $O(1)$ possible values). To find the $(k-j)$-tuples from the right group that sum up to $h(t) - v_l$ a recursive call is done on the arrays of the right group (using their hashed version) where the target value is $h(t) - v_l$. A crucial point is that the algorithm allows the recursive call to return at most $n^{T_{j-2}+1}$ answers. For each answer that we get back from the recursion, we check, in the lookup table T, if the *original* values of the elements in this answer can be completed to a solution that sums up to the target value t. The number of answers the algorithm returns is at most num which in this case is $n^{T_{j-1}+1}$. If there are more answers than num the algorithm returns (to the previous level in the recursion).

In order for this algorithm to work, Wang uses a preliminary Monte Carlo procedure that given an instance of kSUM creates $O(\log n)$ instances of kSUM such that if the original instance has no solution none of these instances has a solution and if it has a solution at least one of these instances has a solution but no more than $O(1)$ solutions. The guarantee that there are at most $O(1)$ solutions is needed to ensure that each recursive call is expected to return the right number of solutions. For example, if the algorithm does a recursive call as explained before on $k - j = T_j + 1 - j = T_{j-1} + 1$ arrays, then we expect that for each value of $h(t) - v_l$ out of the $\Theta(n^{j-1})$ possible values, at most $O((T_{j-1} + 1)/n^{j-1}) = O(n^{T_{j-2}+1})$ answers will be returned from the recursive call. This is because of the almost-balanced property of the hash function. However, if there are many $(k - j)$-tuples whose sum is equal (in their original values), then they will be mapped to the same hash value due to the linearity property of the hash function (to be more accurate, as our hash function is almost-linear there are $O(1)$ possible values that these elements can be mapped to). This is where Wang uses the fact that the new instance of kSUM has no more than $O(1)$ solutions. The number of answers that is returned from the recursive call can be limited to the expected value, as there are at most $O(1)$ $(k - j)$-tuples that have equal sum and are part of a solution because each one of these sums forms a different solution to our kSUM instance and there are at most $O(1)$ such solutions.

We now explain how to modify this algorithm in order to make it a Las Vegas algorithm. The idea is to use the tools we presented for our general self-reduction. This is done in the following theorem:

▶ **Theorem 3.** *For $k \geq 2$, kSUM can be solved by a Las Vegas algorithm that runs in $O(n^{k-\delta f(k)})$ time using $O(n^\delta)$ space, for $0 \leq \delta \leq 1$ and $f(k) \geq \sqrt{2k} - 2$*

Proof. We begin with the algorithm by Wang. The first modification to the algorithm is not to limit the number of answers returned from the recursive call. Let us look at some point in the recursion for which we have j arrays in the left group and $k' - j$ in the right group where the total number of arrays is $k' = T_j + 1$. Wang limited the total number of answers we receive from each of the n^{j-1} recursive calls to be $n^{k'-j}/n^{j-1} = n^{T_j - j+1}/n^{j-1} = n^{T_{j-2}+1}$. This is the expected number of answers we expect to get using a balanced hash function where we do not expect to have many duplicate identical sums. However, even if we do not limit the number of answers we get back from a recursive call the total number of answers we receive back from all the n^{j-1} recursive calls is at most $n^{T_{j-1}+1}$. This is simply because the number of arrays in the right group is T_{j-1+1}. As there can be duplicate sums in this right group the number of answers that we receive from each recursive call (out of the $\Theta(n^{j-1})$ recursive calls) can be much larger than the number of answers we get from another recursive call. Nevertheless, the total number of answers is bounded by the same number as in Wang's algorithm. Now, considering the left group, for every possible value of $v_l \in \Theta(n^{j-1})$ we expect the number of j-tuples that are their hashed sum is v_l to be $O(n)$. This is true unless we have many equal sums that, as explained before, are all mapped to the same value by h. In order to ensure that we save only $O(n)$ j-tuples in the lookup table T, we use our "removing duplicate sums" process. That is, for each j-tuple that is mapped by h to some specific v_l we ensure that there is no previous j-tuple in T that has the same sum (considering the original values of the elements).

Following this modification of the algorithm, we have that the left group is balanced as we expect no more than $O(n)$ entries in T for each possible value of v_l, while the right group may not be balanced. However, what is important is that one group is balanced and the total number of potential solutions in the other groups is the same as in the balanced case. Therefore, we can apply here our "small space of fixed group" observation (see Section 3) that guarantees the desired running time. Verifying each of the answers we get from the right group can be done using our lookup table in $O(1)$ time. Since we have removed duplicate sums (using original values) the expected number of elements that can complete an answer from the right group to a solution to the original kSUM instance is no more than $O(1)$. This is because the number of elements mapped to some specific value of h and having the same value by some h' from some upper level of our recursion is not expected to be more than $O(1)$, as the range of h' is at least n^j and the number of j-tuples is n^j. Therefore, the total running time will be $O(n^{T_{j-1}+1})$ even for our modified algorithm. Moreover, the expected number of answers that are returned by the algorithm for a specific target value is $O(n^{T_{j-2}+1})$.

We note that the answers that are returned from the right group are returned following the "paused reporting" scheme we have described in our self-reduction. We get answers one by one by going back and forth in our recursion and pausing the execution each time we get a candidate solution (it seems that it is also needed in Wang's algorithm though it was not explicitly mentioned in his description).

To conclude, by modifying Wang's algorithm so that the number of the answers returned to the previous level of recursion is not limited and by removing duplicates in the right group (within every level of recursion) we obtained a Las Vegas algorithm that solves (T_j+1)SUM in $O(n^{T_{j-1}+1})$ time and linear space. Using the self-reduction with $m=1$ we have a Las Vegas algorithm that solves kSUM using $O(n^{k-\delta f(k)})$ time, for $f(k) \geq \sqrt{2k}-2$, and $O(n^{\delta})$ space, for $0 \leq \delta \leq 1$. ◀

This Las Vegas algorithm has a better running time than an algorithm using the self-reduction directly because of the additional $\sqrt{2}$ factor before the $-\sqrt{k}$ in the exponent. However, as we will explain in the following sections, there are other uses of our general self-reduction approach.

6 Space-Time Tradeoffs for Large Space

We now consider how to solve kSUM for the case where we can use space which is $O(n^{\delta})$ for $\delta > 1$. We have two approaches to handle this case. The first one is a generalization of the Las Vegas algorithm from the previous section. The second uses our general self-reduction approach from Section 3.

We begin with the first solution and obtain the following result:

▶ **Lemma 6.** *For $k \geq 2$, kSUM can be solved by a Las Vegas algorithm that runs in $O(n^{k-\sqrt{\delta}f(k)})$ time using $O(n^{\delta})$ space, for integer $\frac{k}{4} \geq \delta > 1$ and $f(k) \geq \sqrt{2k}-2$.*

Proof. The proof appears in the full version of this paper. ◀

The approach of the last theorem has one drawback - it gives no solution for the case where we can use $O(n^{\delta})$ space for non integer $\delta > 1$. To solve this case we use our general self-reduction approach and obtain the following:

▶ **Theorem 4.** *For $k \geq 2$, kSUM can be solved by a Las Vegas algorithm that runs in $O(n^{k-\sqrt{\delta}f(k)})$ time using $O(n^{\delta})$ space, for $\frac{k}{4} \geq \delta > 1$ and $f(k) \geq \sqrt{2k}-2$.*

Proof. Recall that the idea of the self-reduction is to split our kSUM instance into k/m groups of m arrays. An almost-linear almost-balanced hash function h is applied to all elements. Then, each group is solved recursively and the answers reported by all of these k/m mSUM instances form an instance of $(\frac{k}{m})$SUM. This approach leads to the following recursive runtime formula: $T(k,n,n^{\delta}) = n^{(k/m-1)(m-\delta)}(T(m,n,n^{\delta}) + T(k/m,n^{\delta},n^{\delta}))$ (see the full details in Section 3). This approach works even for $\delta > 1$. It turns out that the best choice of m for $\delta > 1$ is $m = \lceil \delta \rceil$, which coincides with our choice of m for

$\delta \leq 1$. Following this choice, we have that $T(k, n, n^\delta) = n^{(k/\lceil\delta\rceil-1)(\lceil\delta\rceil-\delta)}(T(\lceil\delta\rceil, n, n^\delta) + T(k/\lceil\delta\rceil, n^\delta, n^\delta)) = O(n^{(k/\lceil\delta\rceil-1)(\lceil\delta\rceil-\delta)}(n^{\lceil\delta\rceil} + T(k/\lceil\delta\rceil, n^\delta, n^\delta)))$. If we plug in our Las Vegas solution following Wang's approach from Section 5, we get that the running time is approximately $O(n^{(k/\lceil\delta\rceil-1)(\lceil\delta\rceil-\delta)}(n^{\lceil\delta\rceil} + n^{\delta(k/\lceil\delta\rceil)-\delta\sqrt{2k/\lceil\delta\rceil}}))$. Therefore, the running time to solve kSUM using $O(n^\delta)$ space for $\delta > 1$ is $O(n^{k-\sqrt{\delta}f(k)})$ for $f(k) \geq \sqrt{2k} - 2$. ◀

We see that for integer values of δ the last result coincides with the previous approach. Using the self-reduction approach even for non integer values of δ, we get similar running time behaviour. We note that using the same ideas from Theorem 3 and the results from Section 4 we can have the same result as Theorem 3 for a deterministic algorithm but with running time which is $O(n^{k-\sqrt{\delta}f(k)})$ for $f(k) \geq \sqrt{k} - 2$.

7 Space Efficient Solutions to 6SUM

In this section we present some space efficient solutions to 6SUM that demonstrate the usefulness of our general self-reduction in concrete cases. For 3SUM we do not know any truly subquadratic time solution and for 4SUM we have an $O(n^2)$ time solution using linear space, which seems to be optimal. Investigating 6SUM is interesting because in some sense 6SUM can be viewed as a problem that has some of the flavour of both 3SUM and 4SUM, which is related to the fact that 2 and 3 are the factors of 6. Specifically, 6SUM has a trivial solution running in $O(n^3)$ time using $O(n^3)$ space. However, when only $O(n)$ space is allowed 6SUM can be solved in $O(n^4)$ time by Wang's algorithm. More generally, using Wang's solution 6SUM can be solved using $O(n^\delta)$ space in $O(n^{5-\delta})$ time for any $\delta \leq 1$. As one can see, on the one hand 6SUM can be solved in $O(n^3)$ time that seems to be optimal, which is similar to 4SUM. On the other hand, when using at most linear space no $O(n^{4-\epsilon})$ solution is known for any $\epsilon > 0$, which has some flavour of the hardness of 3SUM.

There are two interesting questions following this situation: (i) Can 6SUM be solved in $O(n^3)$ time using less space than $O(n^3)$? (ii) Can 6SUM be solved in $O(n^4)$ time using truly sublinear space?. Using our techniques we provide a positive answer to both questions.

We begin with an algorithm that answer the first question and obtain the following result:

▶ **Theorem 7.** *There is a Las Vegas algorithm that solves 6SUM and runs in $O(n^{5-\delta} + n^3)$ time using $O(n^\delta)$ space, for any $\delta \geq 0.5$.*

Proof. The proof appears in the full version of this paper. ◀

By the last theorem we get a tradeoff between time and space which demonstrates in one extreme that 6SUM can be solved in $O(n^3)$ time using $O(n^2)$ space instead of the $O(n^3)$ space of the trivial solution.

The algorithm from the previous theorem runs in $O(n^4)$ time while using $O(n)$ space, this is exactly the complexity of Wang's algorithm for 6SUM. We now present an algorithm that runs in $O(n^4)$ time but uses truly sublinear space.

▶ **Theorem 8.** *There is a Las Vegas algorithm that solves 6SUM and runs in $O(n^{6-3\delta} + n^4)$ time using $O(n^\delta)$ space, for any $\delta \geq 0$.*

Proof. The proof appears in the full version of this paper. ◀

By setting $\delta = 2/3$ in the last theorem, we have an algorithm that solves 6SUM in $O(n^4)$ time while using just $O(n^{2/3})$ space.

References

1 Amir Abboud and Kevin Lewi. Exact weight subgraphs and the k-sum conjecture. In *International Colloquium on Automata, Languages and Programming, ICALP 2013*, pages 1–12, 2013.

2 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Foundations of Computer Science, FOCS 2014*, pages 434–443, 2014.

3 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *International Colloquium on Automata, Languages and Programming, ICALP 2014*, pages 39–51, 2014.

4 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *Symposium on Theory of Computing, STOC 2015*, pages 41–50, 2015.

5 Amihood Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. In *International Colloquium on Automata, Languages and Programming, ICALP 2014*, pages 114–125, 2014.

6 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jussi Määttä. Space-time tradeoffs for subset sum: An improved worst case algorithm. In *International Colloquium on Automata, Languages, and Programming, ICALP 2013*, pages 45–56, 2013.

7 Nikhil Bansal, Shashwat Garg, Jesper Nederlof, and Nikhil Vyas. Faster space-efficient algorithms for subset sum and k-sum. In *Symposium on Theory of Computing, STOC 2017*, pages 198–209, 2017.

8 Ilya Baran, Erik D. Demaine, and Mihai Patrascu. Subquadratic algorithms for 3SUM. In *Workshop on Algorithms and Data Structures, WADS 2005*, pages 409–421, 2005.

9 Gill Barequet and Sariel Har-Peled. Polygon-containment and translational min-hausdorff-distance between segment sets are 3sum-hard. In *Symposium on Discrete Algorithms, SODA 1999*, pages 862–863, 1999.

10 Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In *Theory and Applications of Cryptographic Techniques, EUROCRYPT 2011*, pages 364–385, 2011.

11 Martin Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Symposium on Theoretical Aspects of Computer Science, STACS 1996*, pages 569–580, 1996.

12 Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In *Cryptology Conference, CRYPTO 2012*, pages 719–740, 2012.

13 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995.

14 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. How hard is it to find (honest) witnesses? In *European Symposium on Algorithms, ESA 2016*, pages 45:1–45:16, 2016.

15 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In *Algorithms and Data Structures Symposium, WADS 2017*, pages 421–436, 2017.

16 Isaac Goldstein, Moshe Lewenstein, and Ely Porat. Orthogonal vectors indexing. In *International Symposium on Algorithms and Computation, ISAAC 2017*, pages 40:1–40:12, 2017.

17 Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In *Theory and Applications of Cryptographic Techniques, EUROCRYPT 2010*, pages 235–256, 2010.

18 Richard M. Karp. Reducibility among combinatorial problems. In *Symposium on the Complexity of Computer Computations*, pages 85–103, 1972.

19 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In *Symposium on Discrete Algorithms, SODA 2016*, pages 1272–1287, 2016.

20 Andrea Lincoln, Virginia Vassilevska Williams, Joshua R. Wang, and R. Ryan Williams. Deterministic time-space trade-offs for k-sum. In *International Colloquium on Automata, Languages, and Programming, ICALP 2016*, pages 58:1–58:14, 2016.

21 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Symposium on Theory of Computing, STOC 2010*, pages 603–610, 2010.

22 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In *Symposium on Discrete Algorithms, SODA 2010*, pages 1065–1075, 2010.

23 Richard Schroeppel and Adi Shamir. A T sˆ2 = o(2ˆn) time/space tradeoff for certain np-complete problems. In *Foundations of Computer Science, FOCS 1979*, pages 328–336, 1979.

24 Joshua R. Wang. Space-efficient randomized algorithms for K-SUM. In *European Symposium on Algorithms, ESA 2014*, pages 810–829, 2014.

Dynamic Trees with Almost-Optimal Access Cost

Mordecai Golin
Hong Kong University of Science and Technology
golin@cse.ust.hk

John Iacono[1]
Université libre de Bruxelles and New York University
johniacono@gmail.com

Stefan Langerman[2]
Université libre de Bruxelles
sl@slef.org

J. Ian Munro
Cheriton School of Computer Science, University of Waterloo
imunro@uwaterloo.ca

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

Abstract

An optimal binary search tree for an access sequence on elements is a static tree that minimizes the total search cost. Constructing perfectly optimal binary search trees is expensive so the most efficient algorithms construct *almost optimal* search trees. There exists a long literature of constructing almost optimal search trees *dynamically*, i.e., when the access pattern is not known in advance. All of these trees, e.g., splay trees and treaps, provide a *multiplicative* approximation to the optimal search cost.

In this paper we show how to maintain an almost optimal weighted binary search tree under access operations and insertions of new elements where the approximation is an *additive* constant. More technically, we maintain a tree in which the depth of the leaf holding an element e_i does not exceed $\min(\log(W/w_i), \log n) + O(1)$ where w_i is the number of times e_i was accessed and W is the total length of the access sequence.

Our techniques can also be used to encode a sequence of m symbols with a dynamic alphabetic code in $O(m)$ time so that the encoding length is bounded by $m(H+O(1))$, where H is the entropy of the sequence. This is the first efficient algorithm for adaptive alphabetic coding that runs in constant time per symbol.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Data Structures, Binary Search Trees, Adaptive Alphabetic Coding

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.38

Related Version A full version of the paper is available at [15], `https://arxiv.org/abs/1806.10498`.

[1] Supported by NSF grants CCF-1319648, CCF-1533564, a Fulbright Fellowship, and by the Fonds de la Recherche Scientifique-FNRS under Grant no MISU F 6001 1.
[2] Directeur de recherches du Fonds de la Recherche Scientifique-FNRS.

© Mordecai Golin, Stefan Langerman, John Iacono, J. Ian Munro, and Yakov Nekrich; licensed under Creative Commons License CC-BY

1 Introduction

The dictionary problem is one of the most fundamental problems in computer science. It requires maintaining a set of elements in a data structure and being able to efficiently search for and find them when needed. In the comparison model, balanced binary search trees (BSTs) provide an optimal worst case solution for this problem. We consider leaf-oriented binary search trees, where all of the data is located in leaves and internal nodes store keys needed to guide the search to the leaves. For a set of n elements, it is well known that the perfectly balanced search tree has height $\lceil \log(n+1) \rceil$ and $\lceil \log(n+1) \rceil$ comparisons[3] are required to access an element, both in the worst and average cases. In many practical applications, some elements are known to be accessed more frequently than others; unbalancing and restructuring the tree so that more frequently accessed elements are stored higher up, can lead to better search times. Let d_i be the depth of the i^{th} element e_i (stored at a leaf), w_i the frequency of accessing that element and $W = \sum_i w_i$ the total number of accesses. The total access cost is $\sum_i d_i w_i$; normalizing gives the *tree cost* which is $\frac{1}{W} \sum_i d_i w_i$. A tree that minimizes the tree cost minimizes the total access cost and is an *optimal* BST.

There is a long literature on constructing optimal BSTs, both exactly and approximately[4]. In the approximate case, there are algorithms that provide both multiplicative and additive errors. In the dynamic version of the problem the frequencies w_i are not known in advance but are calculated cumulatively as accesses are made. The problem then is to update the tree to be optimal for the current observed frequencies. Surprisingly, while there are many results on dynamic approximately optimal BSTs with constant multiplicative-error, prior to this paper there was not much known about constant additive-errors.

In this paper we revisit this problem and describe how to maintain dynamic approximately optimal BSTs with constant *additive*-error (this will be formally defined in the next subsection). The cost of re-building the tree after an access operation is bounded by $O(\log^{(f)} n)$ for any constant f, with the additive error growing linearly with f. As in standard BSTs, our technique permits insertions of new elements to the dictionary at any time.

A variant of our approach can also be used to obtain an almost-optimal adaptive alphabetic code with $O(1)$ encoding cost.

Previous and Related Work

There are a number of data structures that maintain (unweighted) dynamic trees with $O(\log n)$ depth, starting with the classic balanced trees of Adelson-Velski and Landis [1] and other handbook solutions [7, 16]. These data structures maintain all leaves at height $O(\log n)$ and thus support both searches and updates, i.e., insertions and deletions, in $O(\log n)$ time. The k-neighbor tree of Maurer et al. [22] achieves tree depth $(1+\delta)\log n$ and update cost $O((1/\delta)\log n)$ for any positive $\delta > 0$. Andersson [4] improved this result and showed how to maintain a tree of height $\log n + O(k)$ in $O(\log n)$ time per update. Even tighter bounds on constant and improved update times were described by Andersson and Lai [6] and Fagerberg [11]. We refer to [5] for an extensive survey of results in this area.

Gilbert and Moore [14] introduced an $O(n^3)$ time algorithm for constructing optimal BSTs. This was improved in 1971 by Knuth [21] to $O(n^2)$, which is still the best known method for solving the general case of the problem. Those two algorithms assume that frequencies for

[3] Throughout this paper log denotes the binary logarithm and $\log^{(f)}$ is the log function iterated f times.
[4] In this paper the term "optimal" refers to the optimality of the tree with respect to access frequencies. Splay trees, for example, can utilize other features of the access sequence in addition to frequencies.

both successful (elements in the tree) and *unsuccessful* (not in the tree) searches are given in advance and optimize accordingly. If the problem is restricted to successful searches then optimal BSTs can be constructed in in $O(n \log n)$ time using the Hu-Tucker algorithm and its variants. [13, 18]. Klawe and Mumey [19] show that, under some general conditions as to how the algorithms can operate, $\Omega(n \log n)$ is the best possible construction time, although, for certain restricted types of input, $O(n)$ can be achieved [17, 19].

Let $p_i = w_i/W$ be the empirical probability of element i in the access sequence. The Shannon Entropy of the sequence is $H = \sum_i p_i \log(1/p_i)$ which is known to be a lower bound on the cost of tree in which all data is in the leaves[5]. If a tree was guaranteed to have $d_i \leq c + \log(1/p_i)$ for all i then the total cost of all accesses would be at most $\sum_i w_i(c + \log(1/p_i)) = WH + cW$, i.e., within a constant additive error per access. In the static case multiple authors [2, 23, 28] have provided $O(n)$ time algorithms for constructing such trees with $c = 2$.

Now consider the dynamic case, in which trees are rebuilt based on cumulative frequencies viewed so far. *Splay trees* [25] and *Treaps,* [24] maintain *static optimality*, essentially keeping element e_i at depth $d_i = O(\log(1/p_i))$ for the current cumulative frequencies, in the amortized sense. This guarantees constant *multiplicative* errors in the dynamic case. There was no comparable result for maintaining almost optimal trees with additive errors, i.e., keeping element e_i at depth $d_i = \log(1/p_i) + O(1)$. The best technique would be to rebuild the tree from scratch at every step.

The dynamic (or adaptive) alphabetic coding problem is closely related to the dynamic alphabetic tree problem just described. The coding problem is to produce an encoding for a sequence of symbols $S[1] \ldots S[m]$ over an ordered alphabet $\{ a_1, \ldots, a_n \}$ so that (1) no codeword is a prefix of any other and (2) the codeword for a_i is lexicographically smaller than the codeword for a_j iff $a_i < a_j$. In the adaptive scenario the input sequence is not known in advance; hence, we need to update the code every time a symbol is encoded. Dynamic Huffman [20, 26] and dynamic Shannon [12] algorithms solve this problem for the non-alphabetic case. The algorithm of Gagie [12] maintains a dynamic alphabetic code, such that the total encoding length is bounded by $(H + 2)m$ and runs in $O(m(H + 1))$ time.

Alphabetic coding is related but not equivalent to the alphabetic trees problem. Any alphabetic tree can be transformed into an alphabetic code in a straightforward way. Hence any dynamic alphabetic tree structure provides us with an alphabetic coding method. But this imposes a lower bound on the encoding time: if the code is represented by a tree, then we have to encode the symbols bit-by-bit. Hence any tree-based alphabetic coding method requires $\Omega(mH)$ time to encode the sequence. On the other hand, not every adaptive coding method can be transformed into a method for maintaining an alphabetic tree. For example, the method of Gagie [12] does not store the alphabetic tree and therefore can not be used to implement a dynamic dictionary.

Notation

The *weight* w_ℓ of a leaf node ℓ is the total number of times that an element stored in ℓ was accessed. We assume that every item is accessed at least once so $w_\ell \geq 1$ The weight of an internal node u is the total weight of all leaves in the subtree of u; the weight of a subtree is equal to the weight of its root. The total weight W of a tree T is the weight of its root node, i.e., $W = \sum_\ell w_\ell$ where the sum is taken over all leaves ℓ. This is also the total number of accesses made.

[5] When data can also be kept in internal nodes, as when three-way comparisons are allowed, the lower bound decreases to $H - \log H$ [3].

When necessary we further denote by $w_\ell^{(j)}$ the number of accesses to ℓ during the first j accesses. Thus $W^{(t)} = \sum_\ell w_\ell^{(t)} = t$.

Relation Between Static and Dynamic Optimal Trees

Consider an optimal static binary search tree for a sequence of W accesses to n elements. As previously noted, the average cost of such a tree is at most $H + 2$ where H is the entropy of the access sequence.

▶ **Lemma 1.** *Let $a_1, a_2, \ldots, a_W$ with $a_i \in \{1, 2, \ldots, n\}$ be a length W* access sequence *on the elements, i.e., element e_{a_t} is accessed at time t. Let H be the entropy corresponding to the full access sequence. Then*

$$\sum_{t=1}^{W} \log \frac{t}{\max\left(w_{a_t}^{(t-1)}, 1\right)} \leq W \cdot H + 2W.$$

The proof of this Lemma is straightforward and is therefore deferred to the full version of this paper [15].

Suppose that we could build a tree $T^{(t)}$ such that the depth of e_i *after* access t is $d_i^{(t)} \leq \log \frac{t}{w_i^{(t)}} + c$. The access of a_t at time t would be in the previous tree $T^{(t-1)}$ with cost $d_{a_t}^{(t-1)}$. The only exception to the above is if time t is the first access to e_{a_t}, so it was not already in $T^{(t-1)}$. In that case the access cost would be $d_{a_t}^{(t)}$, the depth of the location into which a_i^t would be inserted. Thus define $d_{a_t}^{(t-1)} = d_{a_t}^{(t)}$. Since $w_{a_t}^{(t-1)} = 0$ and $w_{a_t}^{(t)} = 1$, $d_{a_t}^{(t-1)} = d_{a_t}^{(t)} \leq \log t + c = \log \frac{t}{\max\left(w_{a_t}^{(t-1)}, 1\right)} + c$. The total cost of the accesses would then, from Lemma 1, be

$$\sum_i d_{a_t}^{(t-1)} \leq W \cdot H + (2 + c)W,$$

i.e, within a constant additive error of optimal per access, where optimal defined as the cost with the static optimal tree, is lower-bounded by $W \cdot H$.

Our approach to building almost optimal trees is therefore to maintain such trees $T^{(t)}$ over the access sequences.

Our Results

Let $f \geq 1$ be any fixed integer. In this paper we describe a dynamic tree structure that can be maintained under access operations and insertions. The depth of the leaf that holds e_i is bounded by $\min(\log(W/w_i), \log n) + O(f)$ where w_i is the number of times e_i was accessed so far and W is the total length of the access sequence. Hence we can access any element e_i using at most $\min(\log n, \log(W/w_i)) + O(1)$ comparisons. We can also insert new elements into the tree. When an element is accessed (resp. when a new element is inserted), only $O(\log^{(f)} n)$ worst-case time will be needed to update the tree; this update procedure does not require any comparisons. Thus our data structure enjoys the advantages of both the weighted alphabetic tree and the perfect binary tree. At the same time, the cost of maintaining the data structure is low.

This result is obtained by a combination of two ideas. First, our construction is based on approximate weights of elements instead of exact weights. Second, we maintain an unweighted binary tree T^{s} with leaves "representing" approximate weights. Our dynamic tree is a subtree of T^{s}. We define the approximate weights in Section 3 and describe the tree T^{s} in Section 4.

Next, we show how updates of our data structure can be implemented by leaf insertions in T^{s} in Section 5. We reduce the update cost and make all time bounds worst-case in Sections 6 and 7 respectively.

Our second result, concerns the adaptive alphabetic coding problem. Our method enables us to encode the sequence of m symbols with an adaptive alphabetic code in $O(m)$ time, constant time per symbol (in contrast to $O(m(H+1))$ time in [12]). The length of encoding is bounded by $m(H+1)+O(m)$ bits. Our solution is based on the same approach as our dynamic tree structure, but we employ a different method to maintain the underlying tree. This method is based on the list maintenance problem [8, 9, 27]. The full details of this result are omitted from this extended abstract but are presented in the full version of this paper [15], in section on alphabetic coding.

2 Preliminaries

An efficient solution for the unweighted search tree problem was presented by Maurer et al. [22]. Their data structure, called a *k-neighbor tree*, is a tree of height $(1+\delta)\log n$, where δ denotes an arbitrarily small positive constant. A k-neighbor tree is a binary tree T such that (1) all leaves in T have the same depth and (2) if a node $u \in T$ has only one child, then u has at least one right neighbor (on the same level), and (3) if a node u has l right neighbors, then $\min(k,l)$ nearest right neighbors of u have two children.

Since this will be used later, we give a sketch of the insertion into such a tree below.

When a new leaf x is inserted into the tree, we find the node p such that the x must be inserted below p and call a recursive procedure $\textsc{Insert}(p,x)$. First, we make x a new child of p. If p has two children, the insertion procedure is completed. If p has three children, we look for a neighbor node q of p such that the distance between p and q is at most k and q has only one child. If q is found, we call the procedure $\textsc{Move}(p,q)$. If q is not found, we create a new node p' that has one child; the only child of p' is the leftmost child of p. If p is not the root node, then we call the procedure $\textsc{Insert}(\mathrm{parent}(p),p')$; otherwise we create a new root node r_n and make both p and p' the children of r_n.

The arguments of the procedure $\textsc{Move}(p,q)$ are two neighbor nodes, p and q, such that p has three children and q has only one child. All nodes u between p and q have two children. The procedure is applied to the children of all nodes u between p and q; every child node is shifted by one position to the right or to the left. At the end p, q, and all nodes u have two children. Thus $\textsc{Move}(p,q)$ consists of d shifts, where d is the distance from p to q. Procedure $\textsc{Move}(p,q)$ needs $O(k)$ time because every node shift takes $O(1)$ time. When a new leaf is inserted, we execute $\textsc{Move}(p,q)$ only one time. Excluding the cost of $\textsc{Move}(p,q)$, we spend $O(1)$ time on every tree level. Therefore a new leaf can be inserted into a tree in $O(\log n+k)$ time. A more detailed description of an insertion can be found in [4]. We can delete a leaf using a symmetric procedure.

The height of a k-neighbor tree with n leaves does not exceed $\left\lfloor \frac{\log n}{\log(2-\frac{1}{k+1})}+1 \right\rfloor$. Using the fact that for any $k \geq \log n$ the height of the tree is bounded by $\log n + O(1)$, Andersson [4] showed how, by using an appropriate value of k the tree height can be bounded by height $\log n + 2$ using only $O(\log n + k) = O(\log n)$ time per operation. It is this version of the data structure that we will use later.

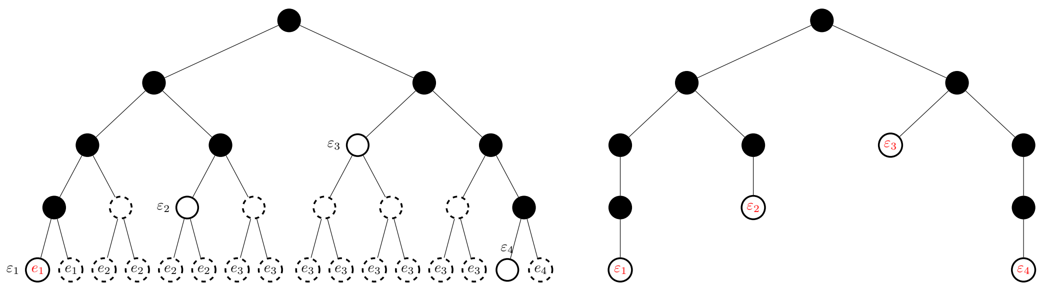


Figure 1 Left: Balanced tree of approximate weights $w'_1 = 1$, $w'_2 = 2$, $w'_3 = 4$, and $w'_4 = 1$. Elements e_1, …, e_4 are stored in nodes ε_1, …, ε_4 respectively. Pseudo-leaves are shown with dashed lines. Internal nodes of T^{s} that are not nodes of T are also drawn with dashed lines. Leaves of T are shown with solid lines and internal nodes of T are depicted by filled circles. Right: Almost-optimal tree corresponding to the tree on Fig. 1.

3 Approximate Weights

Consider an ordered weighted set of elements $E = \{ e_1 < e_2 < \ldots < e_n \}$ let w_i denote the weight of e_i and $W = \sum_{j=1}^{n} w_j$. Define the approximate (or quantized) weight of an element e_i as $w'_i = \lceil w_i/\tau \rceil$ for $\tau = \frac{W}{n}$. Thus all approximate weights are integers between 1 and n. Note that $\sum \frac{w_i}{\tau} = \frac{n}{W} \sum_i w_i = n$. Hence $W' = \sum \lceil \frac{w_i}{\tau} \rceil \le \sum_i \frac{w_i}{\tau} + n = 2n \le 2W$.

▶ **Lemma 2.** *Suppose that the depth of a leaf ℓ_i in a tree T' does not exceed* $\log(W'/w'_i) + c$. *Then the depth of ℓ_i in T' does not exceed* $\min(\log(W/w_i), \log n) + c + 1$.

Proof. Since $w'_i \ge 1$ for all i, $\log(W'/w'_i) \le \log W' \le \log n + 1$. Furthermore $W' \cdot \tau \le 2W$ and $w'_i \cdot \tau \ge w_i$. Hence $\frac{W'}{w'_i} = \frac{W' \cdot \tau}{w'_i \cdot \tau} \le \frac{2W}{w_i}$ and $\log \frac{W'}{w'_i} \le \log \frac{W}{w_i} + 1$.

In summary $\log \frac{W'}{w'_i} \le \min(\log \frac{W}{w_i}, \log n) + 1$. ◀

The problem of maintaining an almost-optimal tree T' for quantized weights $\{ w'_1, \ldots, w'_n \}$ is thus equivalent to the problem of maintaining an almost-optimal tree for exact weights $\{ w_1, \ldots, w_n \}$. The tree T' has another important property: the depths of all leaves in T' are bounded by $\lceil \log n \rceil + O(1)$.

4 Warm-Up: Almost-Optimal Static Trees

In this section we introduce our approach and basic notions that will be used in the following sections. By way of introduction we describe a method that produces an almost-optimal tree for a static set of elements with fixed weights.

We keep weights of elements as entries in an array B of size $m = 2W' \le 2n$ so that there are two entries for each unit of weight. The first $2w'_1$ entries of B are assigned to e_1, the following $2w'_2$ entries are assigned to e_2, and so on. In general we assign entries $B[l_i]$, …, $B[r_i]$ to the element e_i where $l_i = (2\sum_{j=1}^{i-1} w'_i) + 1$ and $r_i = 2\sum_{j=1}^{i} w'_i$. Let T^{s} denote a conceptual perfectly balanced tree on B. The i-th leaf of T^{s} corresponds to the entry $B[i]$ of B, every internal node has two children, and the height of T is $\log m = \log n + 1$[6]. The leaves of T^{s} will be called *pseudo-leaves*. Leaves corresponding to entries in $B[l_i..r_i]$ will be called *pseudo-leaves of the element e_i* (or pseudo-leaves associated to e_i).

[6] To avoid tedious details, we assume in this section that m and n are powers of 2.

▶ **Fact 3.** *Consider a node u of height $h \geq \lfloor \log r \rfloor$ for some $r \geq 1$. Suppose that r leftmost (rightmost) pseudo-leaves in the subtree of u are pseudo-leaves of e_i. Then there is at least one node v of height $\lfloor \log r \rfloor$ such that all pseudo-leaves in the subtree of v are pseudo-leaves of e_i.*
Consequentially, if $2x$ entries are assigned to some element e_i, then there is at least one node v of height $\lfloor \log x \rfloor$, such that all pseudo-leaves in the subtree of v are assigned to e_i.

We define an almost-optimal tree T as a subtree of T^{s}. Let ε_i denote an arbitrary node of height $\lfloor \log(w'_i) \rfloor$ such that all leaves in the subtree rooted at ε_i are i-nodes. Since we assigned $2w'_i$ pseudoleaves to e_i, such a node ε_i always exists. All pseudoleaves below ε_i correspond to some array entries in $B[l_i..r_i]$. The tree T is a subtree of T^{s} pruned at nodes ε_i. That is, the nodes ε_i are the leaves of T and all proper ancestors of all ε_i are internal nodes of T. We keep keys in the internal nodes of T that can be used for routing.

The depth of the leaf ε_i does not exceed $\log \frac{W'}{w'_i}$ by more than a constant: every leaf of T^{s} has depth at most $\log W' + 1$. The depth of ε_i is then at most

$$\log(W') + 1 - (\log(w'_i) + 1) = \log \frac{W'}{w'_i} + 2 \leq \log \frac{W}{w_i} + 3.$$

Hence each ε_i has an almost-optimal depth in T. In addition T^{s} is a perfectly balanced tree with $2n$ nodes and the depth of any node in T^{s} does not exceed $\log n + 1$. Summing up, the depth of any leaf ε_i that holds the element e_i does not exceed $\min(\log(W/w_i), \log n) + 3$.

An example tree T^{s} and the corresponding almost-optimal tree T are shown on Fig. 1. An interesting property of our method is that the tree T is not necessarily a full tree: it is possible that some internal nodes have only one child. In the following sections we will show how the tree T^{s} can be dynamized.

5 Almost-Optimal Dynamic Trees

Our dynamic data structure maintains a balanced tree T^{s} on a dynamic set B of pseudo-leaves.

This first version of the algorithm will work in phases. A phase will end when the total weight W is increased by a factor of 2 or when the total number of elements is increased by a factor of 2.

Unlike in the previous section, these pseudo leaves are not kept in an array. Instead, T^{s} is maintained as a k-neighbor tree data structure with $k = \log n$ [22] as described in Section 2. This method guarantees that all leaves of T^{s} have the same depth and , since the total number of pseudoleaves can at most double within a phase, the height of the tree is bounded by $\log(4W') + 1 \leq \log n + 4$. An update of T^{s} takes $O(\log^2 n)$ time.

Each phase starts with a correct T^{s} that had just been built from scratch using the approach of Section 4. Set $\bar{\tau} = \tau = \frac{W}{n}$. This value stays constant within the phase.

During a phase, for every element e_i we keep track of its weight w_i and its approximate weight $w'_i = \lceil w_i/\bar{\tau} \rceil$. Note that this implies that during a phase w'_i can be increased (incremented by 1 at a step) but not decreased.

When w'_i is incremented by 1, the tree T^{s} is updated: we identify the rightmost pseudo-leaf ℓ_i associated to e_i and insert two new pseudo-leaves, ℓ_n and ℓ_{n+1}, immediately after ℓ_i. When a new element e_f is inserted into a tree, we insert two new pseudo-leaves, ℓ_f and ℓ_{f+1}, into T^{s}. The leaf ℓ_f is inserted after the leaf ℓ_p, where e_p is the largest element satisfying $e_p < e_f$ and ℓ_p is the rightmost leaf associated to e_p. Every insertion of a pseudo-leaf results in a modification of the tree T^{s}.

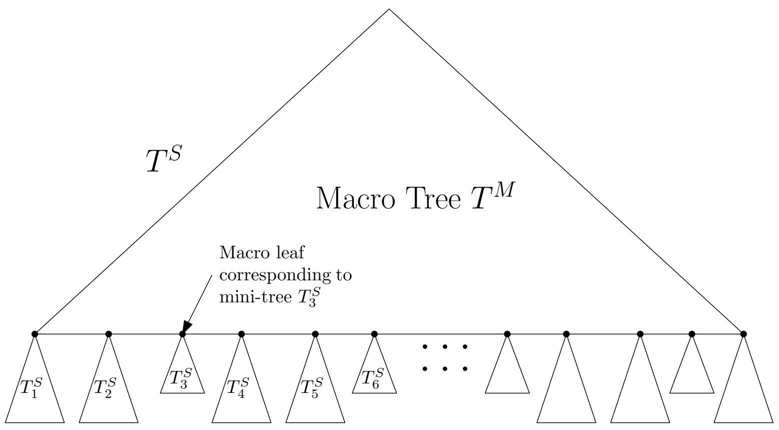


Figure 2 The partition of T^S into macro tree T^M and mini-trees T_j^{s}. The leaves of T^m are the roots of the T_j^{s}. All the T_j^{s} have between $\log^2 n$ and $2\log^2 n$ pseudoleaves. T^M and all of the T_j^{s} are maintained as dynamic almost-optimal trees for their sets of leaves using the technique of Section 5.

We maintain the almost-optimal tree T as a subset of T^{s} using the approach of Section 4. An internal node ε_i is an internal node of T^{s} of height $\lfloor \log(w'_i) \rfloor$ such that all leaves in its subtree are associated to an element e_i. Using the same calculations as in Section 4 the depth of ε_i is then at most $\log \frac{W}{w'_i} + 4$ (and not 3 because the calculation is using $\bar{\tau}$ and not τ.)

After an update of T^{s}, some nodes of T^{s} (and, hence, some nodes of T) can be moved. If all leaves of a moved internal node u are associated to e_j, we also update the internal node ε_j, if necessary. Suppose that a node u was moved by one position to the left and the node u' to the right of u was also moved by one position to the left. If all leaf descendants of u are associated with an element e_i and all leaf descendants of u' are associated with some $e_j \neq e_i$, then we may have to update ε_i. If ε_i is an ancestor of u, we find the immediate left neighbor ε'_i of ε_i. Since there are $2w'_i$ leaves associated to e_i and the height of ε_i is $\log(w'_i)$, all leaf descendants of ε'_i are associated to e_i. Hence we can set $\varepsilon_i := \varepsilon'_i$. We can find the ε_i and ε'_i for every moved node u in $O(\log n)$ time. At most $O(\log n)$ nodes of T^{s} are moved during every update [4]; hence, the total update cost is $O(\log^2 n)$.

When the total weight W is increased by a factor 2 or when the total number of elements is increased by a factor 2, we update the value of $\tau = \frac{W}{n}$, compute the new values w'_i and as noted, re-build the tree from scratch. The amortized cost of rebuilding T^{s} from scratch is $O(1)$ per step since the balanced tree can be built in linear time. When we re-build the tree T^{s}, we use the new value of $k = \log n$.

▶ **Lemma 4.** *We can implement a binary search tree so that access to an element and an insertion of a new element are supported in* $O(\log^2 n)$ *amortized time. If an element e_i was accessed w_i times over a sequence of W operations, then the depth of the leaf holding e_i does not exceed* $\min(\log(W/w_i), \log n) + O(1)$.

6 Faster Updates

We can reduce the update time by grouping pseudo-leaves in the tree T^{s}. All pseudo-leaves are divided into $\Theta(n/\log^2 n)$ groups so that each group contains at least $\log^2 n$ and at most $2\log^2 n$ pseudo-leaves.

The tree T^{s} is divided into two components: a macro-tree $T^M$with $O(n/\log^2 n)$ leaves and $O(n/\log^2 n)$ mini-trees T_j^{s}. See Fig. 2. Mini-trees correspond to groups of pseudo-leaves:

all pseudo-leaves in the group G_j are stored in a mini-tree T^{s}_j. The j'th leaf of macro-tree T^M is the root of mini-tree T^{s}_j. As before, the almost-optimal tree T is a subtree of T^{s}. An element e_i is assigned to a node ε_i of T, such that the height of ε_i in T^{s} is $\log(w'_i)$ (up to an additive constant error) and all leaves in the subtree of ε_i are associated to e_i. T is the subtree of T^{s} induced by nodes ε_i and their ancestors. The division of a tree into macro-trees and mini-trees is a standard data structuring technique; see e.g., [6].

We now find the node ε_i for any element e_i either in a mini-tree or in the macro-tree. Recall that there are $2w'_i$ pseudo-leaves associated to e_i. Let $g = 2\log^2 n$.

First suppose that $w'_i \le g$; then the pseudo-leaves of e_i are distributed among $O(1)$ subtrees. If all pseudo-leaves are in one subtree T^{s}_j, then T^{s}_j has at least one node u of height $\lfloor \log(w'_i)\rfloor$ such that all leaves below u are associated to e_i. If pseudo-leaves of e_i are in two subtrees, T^{s}_j and T^{s}_{j+1}, then either w'_i rightmost pseudo-leaves in T^{s}_j are associated to e_i or w'_i leftmost leaves in T^{s}_{j+1} are associated to e_i. Hence either T^{s}_j or T^{s}_{j+1} contains a node that can be chosen as ε_i. If pseudo-leaves of e_i are distributed among more than two mini-trees, then there is at least one mini-tree T^{s}_j with all pseudo-leaves associated to e_i. In the latter case we can choose the root of T^{s}_j as ε_i.

Now suppose that $kg \le w'_i < (k+1)g$ for some $k \ge 1$. Then there are at least $2k-1$ mini-trees with all pseudo-leaves associated to e_i. The roots of these mini-trees are macro-leaves $\ell_j, \ldots, \ell_{j+2k}$. There is at least one node u of height $\lfloor \log k\rfloor$ in the macro-tree, such that all macro-leaves below u are among $\ell_j, \ldots, \ell_{j+2k}$.

Using Lemma 4, we maintain the mini-tree T^{s}_j for every group G_j. Since each mini-tree has $O(\log^2 n)$ leaves, updates on a mini-tree take $O((\log\log n)^2)$ time. The macro-tree is updated only when a new mini-tree is inserted or a mini-tree is deleted. Hence the cost of updating the macro-tree can be distributed among $O(\log^2 n)$ insertions of pseudo-leaves. Suppose that a new pseudo-leaf corresponding to an element e_i is inserted. As in Section 5 we find the rightmost pseudo-leaf ℓ'_i corresponding to an element e_i. The new pseudo-leaf ℓ_i is inserted into the same mini-tree as ℓ'_i immediately to the right of ℓ'_i. Since every mini-tree has $O(\log^2 n)$ pseudo-leaves, we can insert a new pseudo-leaf in $O((\log\log n)^2)$ time. If the number of pseudo-leaves in T^{s}_j is equal to $2\log^2 n$, we split the mini-tree T^{s}_j into two mini-trees of size $\log^2 n$; then we insert a new macro-leaf into T^{M}. The cost of an insertion into T^{M} is $O(\log^2 n)$. We can also split a mini-tree into two mini-trees in $O(\log^2 n)$ time. Hence the amortized cost of maintaining the macro-tree is $O(1)$.

The total height of a tree does not exceed the height of the macro-tree plus the maximum height of a mini-tree. Since the number of mini-trees is bounded by $\frac{2W'}{(\log^2 n)/2}$, the height of the macro-tree does not exceed $\log(W') - 2\log\log n + 3$. The height of a mini-tree is bounded by $2\log\log n + 1 + O(1)$ because it contains at most $2\log^2 n$ pseudo-leaves. Hence the total height of our tree does not exceed $\log(W') + 4$. We already showed that the height of a sub-tree rooted at the node ε_i is $\lfloor \log(w'_i)\rfloor$; hence the depth of ε_i in T is at most $\log(W'/w'_i) + O(1)$.

▶ **Lemma 5.** *We can implement a binary search tree so that access to an element and an insertion of a new element are supported in $O((\log\log n)^2)$ amortized time. If an element e_i was accessed w_i times over a sequence of W operations, then the depth of the leaf holding e_i does not exceed* $\min(\log(W/w_i), \log n) + O(1)$.

The result of Lemma 5 can be further improved by bootstrapping. For any integer $f \ge 1$ the following statement can be proved.

▶ **Lemma 6.** *Suppose there exists a binary search tree T^f, such that (1) the depth of a leaf holding an element e_i in T^f does not exceed* $\min(\log(W/w_i), \log n) + O(1) + O(f)$ *(2) the amortized cost of updating T^f after an element access or an insertion is $O((\log^{(f)} n)^2)$.*

Then there is a binary search tree T^{f+1}, such that (1) the depth of a leaf holding an element e_i in T^{f+1} does not exceed $\min(\log(W/w_i), \log n) + O(1) + O(f+1)$ *(2) the amortized cost of updating T^{f+1} after an element access or an insertion is $O((\log^{(f+1)} n)^2)$.*

Proof. We divide the tree T^{s} into the macro-tree and mini-trees in the same way as in the proof of Lemma 5. Every mini-tree is implemented using the tree T^f. Hence each mini-tree can be updated in $O((\log^{(f)}(\log n))^2) = O((\log^{(f+1)} n)^2)$ time. The amortized cost of maintaining the macro-tree is $O(1)$. Hence the total amortized cost of updates is $O((\log^{(f+1)} n)^2)$.

Suppose that ε_i is stored in the macro-tree. The depth of a node ε_i in the macro-tree is bounded by $\log(\min(W'/w'_i, n)) + O(1)$. Now suppose that ε_i is stored in some mini-tree. The depth of ε_i in the mini-tree is bounded by $\log(\min(W'_g/w'_i, n_i)) + O(f) + O(1)$, where W'_g is the total sum of all quantized weights in the mini-tree and n_g is the total number of elements in the subtree. By the same argument as in Lemma 5, the depth of ε_i in T is bounded by $\log(\min(W'/w'_i, n)) + O(f+1) + O(1)$. ◀

Our main result is obtained when we apply Lemma 6 $f+1$ times for a parameter $f \geq 0$.

▶ **Theorem 7.** *For any $f \geq 1$ there exists a binary search tree T^f, such that the depth of a leaf holding an element e_i in T^f does not exceed* $\min(\log(W/w_i), \log n) + O(f)$ *and the amortized cost of updating T^f after an element access or an insertion is $O(\log^{(f)} n + f)$.*

We remark that when we insert a new element e_i, we need to update the search path for one leaf. This may incur an additional cost of $\log n + O(1)$ operations.

Our data structure can also support two symmetric operations. We can decrement the weight of an element and delete an element of weight 1. These operations can be implemented in the same way as incrementing the weight of an element and an insertion of a new element.

7 Worst-Case Updates

Our construction can be modified to support updates with worst-case time guarantees. We start by showing how the data structure from Section 5 can be changed. We run several processes in the background; these processes adapt the tree structure to the changing value of the parameter τ and maintain the correct number of pseudo-leaves for each element e_i. The value of τ is changed every time the total weight W for the number of elements is changed by a constant factor (described below). Two background processes guarantee that the value of τ used in T^{s} is within a constant factor of its current value. Moreover pseudo-leaves are stored in a k-neighbor tree data structure, but the parameter $k = \Theta(\log n)$ must be changed when the number of elements is increased or decreased by too much. We run another process that modifies the tree when the parameter k needs to be changed.

Let W_0 and n_0 denote the total weight and the number of elements at some time t_0. Let $\tau_0 = W_0/n_0$ and let the *delayed weight* of an element e_i be defined as $\overline{w}_i = \lceil w_i/\tau_0 \rceil$. We maintain the invariant that w'_i differs from $\overline{w}_i$ by at most a constant factor. In the worst-case construction delayed weights $\overline{w}_i$ are used instead of w'_i, i.e., an element e_i is assigned $\overline{w}_i$ pseudo-leaves. Our re-building processes guarantee that $W_0 \leq W \leq (4/3)W_0$ and $n_0 \leq n \leq (4/3)n_0$. Therefore $\tau = (W/n) \leq (4/3)\tau_0$ and $\tau \geq (3/4)\tau_0$. For any element e_i, $\overline{w}_i = \frac{w_i}{\tau_0} \leq (4/3)w'_i$ and $\overline{w}_i \geq (3/4)w'_i$. Thus we have $\frac{\overline{W}}{\overline{w}_i} \leq (16/9)\frac{W'}{w'_i}$ where $\overline{W} = \sum_i \overline{w}_i$ and $\log(\overline{W}/\overline{w}_i) < \log(W'/w'_i) + 1 \leq \log(W/w_i) + 2$.

We move among three re-building processes. Each process is executed in the background during at most $n/18$ insertions or accesses. The first process updates the value of W_0. If

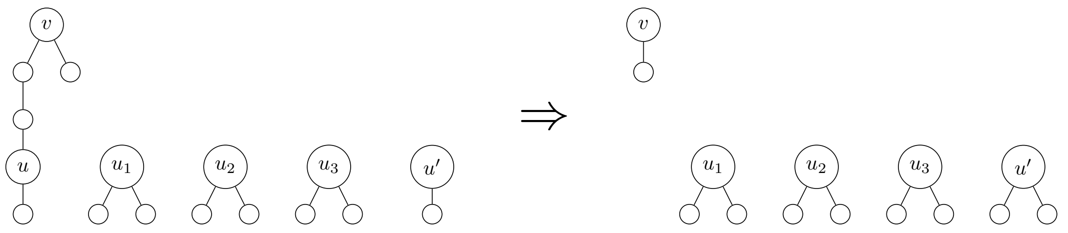


Figure 3 Example of procedure CONSOLIDATE(u, u'). Left: nodes u and u' have one child. Right: node u and its ancestors, up to a node v that has two children, are removed. Children of u_1, u_2, u_3 are shifted one position to the right. Only relevant nodes and their children are shown.

$W \geq (7/6)W_0$, we set $W_1 = W$, $n_1 = n$, and compute $\tau_1 = W_1/n_1$. For every e_i, we compute the new value of $\overline{w}_i = w_i/\tau_1$ and update the tree T^s by removing some pseudo-leaves if necessary. When the number of pseudo-leaves for all elements is adjusted in this way, we set $W_0 = W_1$ and $n_0 = n_1$. The second process updates the value of n_0. If $n \geq (7/6)n_0$, we also compute the new $\tau_1 = W_1/n_1$ for $W_1 = W$ and $n_1 = n$. Then for every element e_i we set $\overline{w}_i = w_i/\tau_1$ and update the tree T^s. The tree always contains $O(n)$ leaves. Every time when we access an element or insert a new element, our background process inserts or removes $O(1)$ pseudo-leaves. We can choose the constant in such a way that adjusting the value of τ_0 is distributed among $n/18$ update or access operations. Suppose that $W \geq (7/6)W_0$ or $n \geq (7/6)n_0$; the value of τ_0 will be adjusted after at most $n/6$ operations. Hence $W \leq (4/3)W_0$ and $n \leq (4/3)n_0$ at any time.

The third background process updates the parameter k in the k-neighbor tree. We set $k_0 = 2(\log(W_0) + 1)$ and maintain a k_0-neighbor tree on pseudo-leaves. When the number of leaves in T^s is increased by factor 2, we start the process of adjusting k. Internal nodes on every level of the tree are divided into pieces, so that every piece consists of $k_0 + 2$ consecutive nodes. We process pieces on the same level in the left-to-right order. Since T^s is already a k_0-neighbor tree, the distance between any two 1-nodes (a 1-node is a node with one child) is at least $k_0 + 1$. Hence each piece contains at most two 1-nodes. If there are two 1-nodes in the same piece P, then they are the leftmost and the rightmost nodes in P. In this case, we execute the procedure CONSOLIDATE(u, u'), where u and u' are the 1-nodes in P. This procedure, that will be described below, removes the node u and adds one additional child to u'. If P contains one 1-node, then we examine the preceding piece P'. If P' also contains a 1-node and the distance between the 1-nodes in P and P' is equal to $k_0 + 2$, we start the procedure CONSOLIDATE(u', u), where u is the 1-node in P and u' is the 1-node in the slide that precedes P . After all pieces on a tree level are processed, every piece contains at most one 1-node and the distance between 1-nodes is at least $k + 2$. We will show below that CONSOLIDATE requires $O(k \log n)$ move operations and can be executed in $O(k \log^2 n) = O(\log^3 n)$ time. Hence the third background process needs $O((n/k) \log^3 n) = O(n \log^2 n)$ time. Since an update takes $O(\log^2 n)$ time, we can distribute the third process among $n/18$ tree updates or accesses.

It remains to describe the procedure CONSOLIDATE(u, u'). CONSOLIDATE(u, u') considers the children of nodes u, u', and the children of all nodes between u and u'. Every such node is moved by one position to the left. As a result, the node has no children and all other considered nodes have two children. Next, let v be the lowest ancestor of u that has two children. The node u and all its ancestors that are below v have no leaf descendants now. We remove the node u and all nodes between v and u. Now the node v is a 1-node. If v is the root node, then we remove v. Otherwise, we check whether v has a neighbor v', such that the distance between v and v' does not exceed $k_0 + 1$ and v' is a 1-node. If v' exists, we recursively call the procedure $\textsc{Consolidate}(v, v')$ (respectively $(\textsc{Consolidate}(v', v))$. There is at most one recursive call of our procedure per tree level. Our procedure shifts $O(k_0)$ nodes by one position to the right and recursively calls itself on some higher tree level; every time when some node in T^{s} is shifted, we may have to move some leaf ε_i of T. Hence the total time of $\textsc{Consolidate}(u, u')$ is $O(k_0 \log^2 n) = O(\log^3 n)$.

▶ **Lemma 8.** *We can implement a binary search tree so that access to an element and an insertion of a new element are supported in* $O(\log^2 n)$ *time. If an element* e_i *was accessed* w_i *times over a sequence of* W *operations, then the depth of the leaf holding* e_i *does not exceed* $\min(\log(W/w_i), \log n) + O(1)$.

7.1 Fast Updates

Now we show how the data structure from Section 6 can be changed to support updates in worst-case time. As in Section 6 the tree T^{s} is divided into the macro-tree and mini-trees. Each mini-tree contains $O(\log^3 n)$ pseudo-leaves.

A new pseudo-leaf is inserted into a mini-tree; the cost of an insertion is $O((\log\log n)^2)$ time by Lemma 8. We run an additional background process that maintains the sizes of mini-trees. During each iteration we identify the largest mini-tree T_l among all subtrees of size at least $(7/4)\log^3 n$. We split T_l into two mini-trees of almost-equal size. We also identify the smallest mini-tree T_k of size at most $(3/4)\log^3 n$; we merge T_k with one of its direct neighbors (i.e., with the mini-tree immediately to the left or immediately to the right of T_k). If the resulting mini-tree is larger than , then we split it into two almost-equal parts. We show in the full version [15] how a mini-tree can be split into two almost-equal parts or merged with another mini-tree in less than $O(\log^2 n)$ time. When we split or merge two mini-trees, we also have to perform $O(1)$ updates on the macro-tree. The cost of updates is $O(\log^2 n)$, hence each iteration takes $O(\log^2 n(\log\log n)^2)$ time. By Theorem 5 from [10], we can organize our background process so that each mini-tree has no more than $2\log^3 n$ and no less than $\log^3 n/2$ pseudo-leaves.

▶ **Lemma 9.** *We can implement a binary search tree so that access to an element and an insertion of a new element are supported in* $O((\log\log n)^2)$ *amortized time. If an element* e_i *was accessed* w_i *times over a sequence of* W *operations, then the depth of the leaf holding* e_i *does not exceed* $\min(\log(W/w_i), \log n) + O(1)$.

We can recursively apply Lemma 9 in the same way as described in Section 6. To obtain the main result of this paper with worst-case guarantees, we apply Lemma 9 $k + 1$ times for a parameter $k > 1$.

▶ **Theorem 10.** *For any* $k \geq 1$ *there exists a binary search tree* T^k*, such that the depth of a leaf holding an element* e_i *in* T^k *does not exceed* $\min(\log(W/w_i), \log n) + O(k)$ *and the cost of updating* T^k *after an element access or an insertion is* $O(\log^{(k)} n + k)$.

References

1 Georgy Adelson-Velsky and Evgenii Landis. An algorithm for the organization of information. *Soviet Mathematics – Doklady*, 3:1259–1262, 1962.

2 Rudolf Ahlswede and Ingo Wegner. *Search Problems.* John Wiley and Sons, Chichester, 1987.

3 Brian Allen. On the costs of optimal and near-optimal binary search trees. *Acta Informatica*, 18:255–263, 1982.

4 Arne Andersson. Optimal bounds on the dictionary problem. In *Proc. International Symposium on Optimal Algorithms*, pages 106–114, 1989.

5 Arne Andersson, Rolf Fagerberg, and Kim S. Larsen. Balanced binary search trees. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004.

6 Arne Andersson and Tony W. Lai. Comparison-efficient and write-optimal searching and sorting. In *(Proc. 2nd International Symposium on Algorithms (ISA '91)*, pages 273–282, 1991.

7 Rudolf Bayer. Symmetric binary b-trees: Data structure and maintenance algorithms. *Acta Informatica*, 1(4):290–306, Dec 1972.

8 Michael A. Bender, Richard Cole, Erik D. Demaine, Martin Farach-Colton, and Jack Zito. Two simplified algorithms for maintaining order in a list. In *Proc. 10th Annual European Symposium on Algorithms (ESA 2002)*, pages 152–164, 2002.

9 Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, Tsvi Kopelowitz, and Pablo Montes. File maintenance: When in doubt, change the layout! In *Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 1503–1522, 2017.

10 Paul Dietz and Daniel Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC 1987)*, pages 365–372. ACM, 1987.

11 Rolf Fagerberg. Binary search trees: How low can you go? In *Proc. 5th Scandinavian Workshop on Algorithm Theory (SWAT '96)*, pages 428–439, 1996.

12 Travis Gagie. Dynamic shannon coding. In *Proc. 12th Annual European Symposium on Algorithms (ESA 2004)*, pages 359–370, 2004.

13 Adriano M. Garsia and Michelle L. Wachs. A New Algorithm for Minimum Cost Binary Trees. *SIAM Journal on Computing*, 6(4):622–642, 1977.

14 E.N. Gilbert and E.F. Moore. Variable-length binary encodings. *Bell System Technical Journal*, 38(4):933–967, 1959.

15 Mordecai Golin, John Iacono, Stefan Langerman, J. Ian Munro, and Yakov Nekrich. Dynamic trees with almost-optimal access cost. `arXiv:1806.10498`.

16 Leonidas J. Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science (FOCS 1978)*, pages 8–21, 1978.

17 T. C. Hu, Lawrence L Larmore, and J David Morgenthaler. Optimal Integer Alphabetic Trees in Linear Time. In *13th Annual European Symposium on Algorithms (ESA'05)*, volume 3669, pages 226–237, 2005.

18 T. C. Hu and A. C. Tucker. Optimal Computer Search Trees and Variable-Length Alphabetical Codes. *SIAM Journal on Applied Mathematics*, 21(4):514–532, 1971.

19 Maria Klawe and Brendan Mumey. Upper and Lower Bounds on Constructing Alphabetic Binary Trees. *SIAM Journal on Discrete Mathematics*, 8(4):638–651, 1995.

20 D. E. Knuth. Dynamic Huffman coding. *Journal of Algorithms*, 6:163–180, 1985.

21 Donald E. Knuth. Optimum binary search trees. *Acta informatica*, 1(1):14–25, 1971.

22 Hermann A. Maurer, Thomas Ottmann, and Hans-Werner Six. Implementing dictionaries using binary trees of very small height. *Information Processing Letters*, 5(1):11–14, 1976.

23 Kurt Mehlhorn. A best possible bound for the weighted path length of binary search trees. *SIAM Journal on Computing*, 6(2):235–239, 1977.

24 Raimund Seidel and Cecilia R. Aragon. Randomized search trees. *Algorithmica*, 16(4):464–497, 1996.

25 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3):652–686, 1985.

26 J. S. Vitter. Design and analysis of dynamic Huffman codes. *Journal of the ACM*, 1987(4):825–845, 1987.

27 Dan E. Willard. A density control algorithm for doing insertions and deletions in a sequentially ordered file in good worst-case time. *Inf. Comput.*, 97(2):150–204, 1992.

28 R.W. Yeung. Alphabetic codes revisited. *IEEE Transactions on Information Theory*, 37(3):564–572, may 1991.

A Tree Structure For Dynamic Facility Location

Gramoz Goranci
University of Vienna, Faculty of Computer Science, Vienna, Austria
gramoz.goranci@univie.ac.at

Monika Henzinger
University of Vienna, Faculty of Computer Science, Vienna, Austria
monika.henzinger@univie.ac.at

Dariusz Leniowski
University of Vienna, Faculty of Computer Science, Vienna, Austria
dariusz.leniowski@univie.ac.at

Abstract

We study the metric facility location problem with client insertions and deletions. This setting differs from the classic dynamic facility location problem, where the set of clients remains the same, but the metric space can change over time. We show a deterministic algorithm that maintains a constant factor approximation to the optimal solution in worst-case time $\tilde{O}(2^{O(\kappa^2)})$ per client insertion or deletion in metric spaces while answering queries about the cost in $O(1)$ time, where κ denotes the doubling dimension of the metric. For metric spaces with bounded doubling dimension, the update time is polylogarithmic in the parameters of the problem.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases facility location, dynamic algorithm, approximation, doubling dimension

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.39

Funding The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506.

1 Introduction

In the *metric facility location problem*, we are given a (possibly infinite) set V of *potential clients* or *points*, a finite set $\mathcal{C}$ of (*live clients*), a finite set $J \subseteq V$ of *facilities* with an *opening cost* f_j, for each facility j, and a metric d over V, such that $\mathrm{d}(i,j)$ is the cost of assigning client i to facility j. The goal is to determine a subset $J' \subseteq J$ of *open* facilities and to assign each client to an open facility such as to minimize the total cost. Obviously it is best to assign each live client to the closest *open* facility. Thus, the goal can be written as minimizing the objective function $\sum_{j\in J'} f_j + \sum_{i\in\mathcal{C}} \min_{j\in J'} \mathrm{d}(i,j)$.

The facility location problem is one of the central problems in combinatorial optimization and operations research [7], with many real-word applications. Typical examples include placements of servers in a network, location planning for medical centers, fire stations, restaurants, etc. From the computational perspective, this problem is NP-hard and it is even hard to approximate to a factor better than 1.463 [13, 20]. The best-known polynomial-time algorithm achieves a 1.488-approximation [17].

In many applications of facility location, problem data are continuously changing. This has lead to the study of this problem in different settings, e.g., online [18, 3, 10, 4, 11, 19, 1], streaming [15, 12, 16, 6] or dynamic [21, 5, 9, 8]. The focus of this paper is on the dynamic

© Gramoz Goranci, Monika Henzinger, and Dariusz Leniowski;
licensed under Creative Commons License CC-BY

setting, motivated by mobile network applications, where the set of clients may change over time, and we need to maintain the set of opened servers so as to obtain a solution of small cost after each change.

Formally, in the *dynamic facility location* problem, the set of clients $\mathcal{C}$ evolves over time and queries about both the cost as well as the set of opened facilities can be asked. Specifically, at each timestep t, either a new client is added to $\mathcal{C}$, a client is removed from $\mathcal{C}$, a query is made for the approximate cost of an optimal solution (*cost query*), or a query asks for the entire current solution (*solution query*). The goal is to maintain a set of open facilities that after each client update minimizes the above cost function. Thus, the cost f_j of each facility can be seen as a *maintenance cost* that has to be paid for each open facility between two client updates.

Our contribution. In this paper we present a *deterministic* data-structure that maintains a $O(1)$-factor approximation algorithm for the metric facility location problem, while supporting insertions and deletions of clients in $\tilde{O}(2^{O(\kappa^2)})$ update time, and answering cost queries in $O(1)$ time, where κ is the *doubling dimension*[1] of the metric space. As the running time per client update is bounded by $\tilde{O}(2^{O(\kappa^2)})$, the number of changes in the client-facility assignments is also bounded by this function. For metric spaces with bounded doubling dimension, such as the Euclidean space, the running time is $\tilde{O}(1)$. Formally, we have the following theorem.

▶ **Theorem 1.** *There exists a deterministic algorithm for the dynamic facility location problem where clients and facilities live in a metric space with doubling dimension κ, such that at every time step the solution has cost at most $O(1)$ times the cost of an optimal solution at that time. The worst-case update time for client insertion or deletion is $O(2^{O(\kappa^2)} \cdot \Delta^3 \cdot (\kappa^2 + \log \Delta))$, where Δ is logarithmic in the paramters of the problem. A cost query can be answered in constant time and a solution query in time linear in the size of the output.*

Comparison with prior work. The closest work related to our problem is the *streaming* algorithm for the *metric* facility location problem with *uniform* opening costs due to Lammersen and Sohler [16]. Specifically, given a sequence of insert and deletion operations of points (clients) from $\{1, \ldots, \Delta\}^d$, they devise a Monte-Carlo *randomized* algorithm that processes an insertion or deletion of a point in $\tilde{O}(2^{O(d)})$ time, using poly-logarithmic space and maintaining a $\tilde{O}(2^{O(d)})$-factor approximation. Since the d-dimensional Euclidean space has doubling dimension linear in d, we can also interpret the above result in terms of κ, i.e., the same bounds hold with d replaced by κ. An easy inspection of the algorithm in [16] shows that the queries can also be answered anytime in the update sequence in $O(1)$ time. Note that this algorithm heavily relies on randomization and the fact that facilities have uniform opening costs. In comparison our algorithm is (1) *deterministic*, (2) achieves a $O(1)$-factor approximation (independent of doubling dimension) (3) generalizes to any metric of bounded doubling dimension, and (4) supports non-uniform opening costs. Furthermore, for the Euclidean plane, i.e. $d = 2$, there is a randomized streaming algorithm that achieves a $(1 + \varepsilon)$-approximation with poly-logarithmic space [6]. However, it is not clear whether this algorithm supports fast queries.

Regarding the *dynamic facility location problem*, multiple variants can be found in the literature [21, 5, 9, 8]. All these variants are different from ours as they assume that the

[1] The doubling dimension of a metric space (V, d) is bounded by κ if for any $x \in V$ and any radius r, any ball with center x and radius r in (V, d) can be completely covered by 2^κ balls of radius $r/2$.

facilities and clients remain the same, and only *the distance metric* between clients and facilities can change. Additionally, every time a client switches to a different facility, a switching cost might be incurred and the goal is to minimize the above sum plus all the switching costs. For the version proposed in [8], there exists a polynomial time constant factor approximation algorithm [2].

There is also a large body of prior work on *online facility location* (see, e.g., [18, 3, 10, 4, 11, 19, 1]), where the clients arrive in an online fashion and have to be connected to a facility, potentially opening new facilities. If earlier decisions cannot be reversed, then no efficient constant factor approximation algorithm is possible [11]. If earlier decisions are, however, not permanent, specifically if facilities are only opened for a given amount of time, i.e. *leased*, and k different lease lengths are possible [4], the offline version of the problem has a polynomial time 3-approximation algorithm [19], but the online setting cannot have an approximation better than $\Omega(\log k)$, even for randomized algorithms [18]. All of these results are different from ours: (1) We have only one lease length, namely length 1, as each facility can be closed or opened after each timestep, (2) we allow client arrival *and* departure, and (3) our algorithm processes a client update in $O(2^{O(\kappa^2)} \cdot \Delta^3 \cdot (\kappa^2 + \log \Delta))$ worst-case time per operation, where Δ is logarithmic in the parameters of the problem, while the running time of the online algorithms is at least linear in the number of facilities [18, 3].

Technical contribution. From a technical point of view we modify and significantly extend a hierarchical partition of a subset of the facilities that was recently introduced for a related problem, called the *dynamic sum-of-radii clustering* problem [14]. In that work, a set of facilities J and a dynamically changing clients $\mathcal{C}$ are given and the goal is to output a set $J' \subseteq J$ together with a *radius* R_j, for each $j \in J'$, such that the J' covers $\mathcal{C}$ and the function $\sum_{j \in J'}(f_j + R_j)$ is minimized. In [14] a $O(2^{2\kappa})$-approximation algorithm with time $\tilde{O}(2^{6\kappa})$ per client insertion or deletion is presented for metrics with doubling dimension κ. Note that the function that is minimized is different from the function minimized in the facility location problem, as the term $\sum_{i \in \mathcal{C}} \min_{j \in J'} \mathrm{d}(i, j)$ is replaced by $\sum_{j \in J'} R_j$.

More specifically, the hierarchical decomposition of [14] picks a well-separated subset of J with "small" cost, assigns one or multiple radii to the selected facilities, and then hierarchically orders the pairs $\langle j, R \rangle$ in a tree structure, where j is a facility and R is a radius, such that the children of every pair have a smaller radius and the "ball" of the given radius of a child is fully contained in the "ball" of its parent with its radius. To achieve our result, in Sections 2 and 3 we extend this decomposition as follows:

1. *Abundance condition.* Instead of selecting facilities with "small" cost, we introduce the notion of an "abundance condition": Facilities that have "enough nearby" clients fulfill this condition, and we *only open such facilities*. This leads to following rough notion: The abundance condition is fulfilled for a facility j and a radius R if the number of clients within radius R of j is at least f_j/R. The fundamental idea is then as follows: (A) We assign a *payment* of R to each client within radius R of an open facility j, which implies that the sum of f_j plus the distances of these clients to j is upper bounded by twice the sum of the payments of these clients. (B) We then show that (i) each client pays for at most one facility and (ii) the sum of the client payments is linear in the cost of the optimal solution.
2. *Designated facilities.* To further reduce the cost of the open facilities, we designate to each facility j the "cheapest nearby" facility j^*, and if a facility j fulfills all conditions to be opened, we open the facility j^* instead. This allows us to modify the (rough) abundance condition so that it is fulfilled for a facility j and a radius R if the number of

clients within radius R of j is at least f_{j^*}/R. This modification increases the distance of the clients only by a constant factor, but might significantly decrease the cost of the open facilities. Note that our approach would not achieve a constant factor without this technique: The hierarchical decomposition is based on a well-separated subset of facilities whose construction ignores the facility costs. Thus it might happen that the chosen facilities have high cost, even though there are "cheap" facilities nearby. These cheap nearby facilities are now captured by the designated facilities.

3. *Enabled facilities.* The idea of not opening facilities that are "close" to an open facility can be further combined with the hierarchical decomposition. More specifically, if facility j with radius R and facility j' with radius R' are "close", only one of them is opened, namely the lowest-in-the-hierarchy facility that fulfills the abundance condition and has no nearby facility of smaller radius that is already open. The advantage of this scheme is that when a facility switches from open to closed or vice versa, we can bound the number of facilities that are affected by this change by a function that only depends on κ. If we had chosen to open the facility with larger radius, then the number of facilities that are affected by opening or closing one facility might have been large, i.e., not bounded by a function of κ alone.
4. *Coloring.* Recall that we need to guarantee that each client only pays for *one* open facility. To do so, we assign a *color* to each pair $\langle j, r\rangle$ in the hierarchy such that no two pairs $\langle j, r\rangle$ and $\langle j', r\rangle$, where the distance between j and j' is small, have the same color. As the metric has bounded doubling dimension, $2^{5\kappa}+1$ colors suffice for this coloring. Then we require that a facility is opened only if it fulfills the abundance condition, *and* no facility of either smaller radius or the same radius but with "smaller" color is already open. This requires to further relax the notion of "closeness" but reduces the number of open facilities enough so that each client can be assigned to pay for at most one "close" open facility and still every open facility has enough clients paying for its cost.

In Section 3, Theorem 24, we show that (a) for clients that are "close" to an open facility j in the *optimal solution* the sum of their payments (in our solution) is linear in f_j and (b) for clients that are "far" from an open facility in the *optimal solution* their payments (in our solution) are within a constant factor of their distance in the optimal solution. Additionally, we give a data structure that maintains this solution efficiently under insertions and deletion of clients (see Section 4). All missing proofs are deferred to the full version.

2 Preprocessing phase

Let W be the diameter of the metric space, i.e., $\mathrm{d}(i,j) \le W$, for all $i, j \in V$, let $f_{\max} = \max_{j\in J}\{f_j\}$ be the maximum facility opening cost, and let $f_{\min} = \min\{f_j \mid j \in J, f_j > 0\} > 0$ be the minimum opening cost of any facility with non-zero opening cost. This is w.l.o.g. as all facilities with 0 opening cost are always kept open. Given the set of clients $\mathcal{C} \subseteq V$, let $\mathrm{OPT} = \mathrm{OPT}(\mathcal{C})$ denote the cost of an optimum solution for $\mathcal{C}$. In what follows, for the sake of exposition, we also let $\mathcal{C}$ to refer to the *current* set of clients.

The algorithm will maintain a number $n = 5^{\lfloor \log_5 |\mathcal{C}| \rfloor}$, i.e., the largest power of 5 smaller than $|\mathcal{C}|$: Initally we set n to 0 and use this value of n during preprocessing. Whenever the first client is inserted, we set $n = 1$. Afterwards, whenever the number of clients is a factor 5 larger, resp. smaller, than n, we update n by multiplying, resp. dividing it by 5.

Now let[2] $\rho_{\min} = \lceil \log_5 (f_{\min}/\max(|J|, n)) \rceil$ and let $\rho_{\max} = \lceil \log_5 (\max(W, f_{\max})) \rceil = O(\log W + \log f_{\max})$. Note that a change of $\rho_{\min}$ will require an update in our data structures, but this will only happen after $\Theta(n)$ many client insertions or deletions. As we will see later, the cost of this update will be charged against these client updates, and thus does not affect our running times. A *logradius* is an integer r such that $\rho_{\min} \leq 5^r \leq \rho_{\max}$. Let $\Delta = \rho_{\max} - \rho_{\min} + 1$ be the number of different logradii. Note that $\Delta = O(\log W + \log(f_{\max}/f_{\min}) + \log|J| + \log|\mathcal{C}|)$. Finally, let $c_1 = 20$, $c_2 = 35$, $c_X = 2c_2 + 2 = 72$, $c_3 = c_X + c_2 = 107$, $c_Y = 2c_3 + c_2 = 249$ and $c_4 = c_Y + c_2 = 284$.

A large part of our data structure is concerned with reducing the number of facilities that are potentially opened and finding an assignment of each client to at most one open facility. This is done in multiple ways, as described next. Based on the approach of [14], we construct a set of pairs $\Pi \subseteq (J \times [\rho_{\min}, \rho_{\max}])$, consisting of facility-logradius pairs and a laminar family of *areas*. Different from [14], we color pairs in Π, turning them into triplets, and introduce designated facilities, before defining open, closed and enabled triplets.

Maximal subsets of distant facilities. The first step is to filter out facilities that are close to other facilities. To achieve this, we greedily construct a set Π of *pairs* $\langle j, r\rangle$ where j is a facility and r is a logradius, satisfying the following properties:

1. (Covering) For every facility $j \in J$ and every logradius r, there exists a facility $j' \in J_r$ with $\mathrm{d}(j, j') \leq c_1 \cdot 5^r$.
2. (Separating) For all distinct $j, j' \in J_r$, $\mathrm{d}(j, j') > c_1 \cdot 5^r$.

We construct Π as follows: For each logradius $r \in [\rho_{\min}, \rho_{\max}]$, let J_r be a maximal subset of J such that any two facilities in J_r are at distance strictly larger than $c_1 \cdot 5^r$. Set $\Pi \leftarrow \bigcup_r \{\langle j, r\rangle \mid j \in J_r\}$. Note that for $r = \rho_{\max}$, the set J_r contains just one facility.

Hierarchical decomposition of Π. We now construct a hierarchical decomposition of Π and represent it by a tree $\mathcal{T}$, using the following algorithm. Set the root of $\mathcal{T}$ to be the unique pair $\langle j, \rho_{\max}\rangle$. For each $r < \rho_{\max}$ and $j \in J_r$: (1) Set $j' \in J_{r+1}$ be the facility closest to j. (2) Set $\mathrm{parent}(j, r) \leftarrow \langle j', r+1\rangle$.

By construction, $\mathcal{T}$ has height at most Δ and the parent of a pair $\langle j, r\rangle$ is a pair of the form $\langle j', r+1\rangle$. The following three lemmata describe the crucial properties of the tree $\mathcal{T}$.

▶ **Lemma 2** (Nesting of balls). *Let c^* be any constant such that $c^* \geq (5/4)c_1$. If $\mathrm{parent}(j, r) = \langle j', r+1\rangle$, then $\mathrm{d}(j, j') \leq c_1 \cdot 5^{r+1}$ and $B(j, c^* \cdot 5^r) \subseteq B(j', c^* \cdot 5^{r+1})$.*

▶ **Lemma 3** ([14]). *For any point p, radius r and some number $\alpha > 0$, the set of pairs $\Pi(p, r) = \{\langle j, r\rangle \in \Pi \mid \mathrm{d}(p, j) < 2^\alpha c_1 \cdot 5^r\}$ has at most $2^{(\alpha+1)\kappa}$ elements, where κ is the doubling dimension of the metric space.*

▶ **Lemma 4** ([14]). *A node $\langle j, r\rangle$ of $\mathcal{T}$ has at most $2^{4\kappa}$ children*

Hierarchical decomposition of V into a laminar family of areas. The balls $B(j, r)$ and $B(j', r)$ with $\langle j, r\rangle$ and $\langle j', r\rangle$ in Π might overlap, which is problematic for the mapping of clients to facilities. To rectify this problem, we partition V into a laminar family of *areas* such that no two same-logradius areas overlap, as follows: For each $\langle j, r\rangle \in \Pi$, initialize $A(j, r) \leftarrow \emptyset$. Next, for each point $p \in V$: (1) Let r^* be the smallest such that there exists pairs $\langle j, r^*\rangle \in \Pi$ with $p \in B(j, c_2 \cdot 5^{r^*})$. (2) Among all such pairs, let $\langle j^*, r^*\rangle$ denote the

[2] Note that $\rho_{\min}$ could be negative, but it is well-defined as $f_{\min} > 0$.

one minimizing $\mathrm{d}(p, j^*)$. (3) Add p to the set $A(j^*, r^*)$ and to every set $A(j', r')$ with (j', r') ancestor of (j^*, r^*) in $\mathcal{T}$.

The laminar family of areas fulfills the following lemmata.

▶ **Lemma 5** ([14])**.** *For each $\langle j, r\rangle \in \Pi$, $j \in A(j, r) \subseteq B(j, c_2 \cdot 5^r)$ and if $\mathrm{parent}(j, r) = \langle j', r+1\rangle$, then $A(j, r) \subseteq A(j', r+1)$.*

▶ **Lemma 6.** *Let $j \in J$, $r \in [\rho_{\min}, \rho_{\max}]$ and $p \in B(j, 5^r)$. Then there exists a pair $\langle j', r\rangle \in \Pi$ such that $p \in A(j', r)$ and $\mathrm{d}(j, j') \leq (c_2 + 1) \cdot 5^r$.*

Additionally an even stronger statement regarding the points covered by areas versus points covered by balls holds:

▶ **Lemma 7.** *For each logradius $r \in [\rho_{\min}, \rho_{\max}]$, $\bigcup_{\langle j,r\rangle \in J_r} A(j, r) = \bigcup_{\langle j,r\rangle \in J_r} B(j, c_2 \cdot 5^r)$.*

Covering balls using unions of areas. To select which facilities with a pair $\langle j, r\rangle$ in Π to open, we introduce below the abundance condition which measures how many clients are "close" to j. For measuring "closeness", we would like to say that a client i is close to a facility j if $i \in \mathrm{X}(j, r)$ for some definition of $\mathrm{X}(j, r)$ that fulfills the crucial property that for every $\langle j, r\rangle$ there exists a pair $\langle j', r\rangle \in \Pi$ such that $B(j, 5^r) \subseteq \mathrm{X}(j', r)$. Note that this might not hold if we use $\mathrm{X}(j, r) = A(j, r)$ and it does not follow from Lemma 6 as different points of $B(j, 5^r)$ might belong to different areas $A(j', r)$, whose facilities might be up to distance $(2c_2 + 2) \cdot 5^r$ apart. Thus, for any $\langle j, r\rangle \in \Pi$, we define $\mathrm{X}(j, r)$ as follows:

$$\mathrm{X}(j, r) = \bigcup \{A(j', r) \mid \mathrm{d}(j, j') \leq c_X \cdot 5^r\}, \qquad \text{where } c_X = 2c_2 + 2,$$

and can now show the desired property for $\mathrm{X}(j, r)$:

▶ **Lemma 8.** *Let $j \in J$ with $\langle j, r\rangle \notin \Pi$. Then there exists $\langle j^*, r\rangle \in \Pi$ with $B(j, 5^r) \subseteq \mathrm{X}(j^*, r)$.*

Proof of Lemma 8. It suffices to show that there exists $\langle j^*, r\rangle \in \Pi$ such that for every area $A(j', r)$ that intersects with the ball $B(j, 5^r)$, we get that $A(j', r) \subseteq \mathrm{X}(j, r)$. First, by the Covering property, there exists a pair $\langle j^*, r\rangle \in \Pi$ such that $\mathrm{d}(j, j^*) \leq c_1 \cdot 5^r$. Next, let $A(j', r)$ be any area such that $A(j', r) \cap B(j, 5^r) \neq \emptyset$. By Lemma 6, we get that $\mathrm{d}(j', j) \leq (c_2 + 1) \cdot 5^r$. It follows that $\mathrm{d}(j', j^*) \leq \mathrm{d}(j', j) + \mathrm{d}(j, j^*) \leq (c_1 + c_2 + 1) \cdot 5^r \leq c_X \cdot 5^r$, which in tun implies that $A(j', r) \subseteq \mathrm{X}(j^*, r)$. ◀

It is crucial for the running time to get a bound on the number of areas used to construct $\mathrm{X}(j, r)$. We do this in the following lemma, which is a simple corollary of Lemma 3.

▶ **Lemma 9.** *For any $\langle j, r\rangle \in \Pi$, $\mathrm{X}(j, r)$ is a union of at most $2^{3\kappa}$ areas.*

We also need to bound how far any two points in $\mathrm{X}(j, r)$ can be apart:

▶ **Lemma 10.** *It holds that $\mathrm{X}(j, r) \subseteq B(j, c_3 \cdot 5^r)$.*

To further reduce the set of open facilities, it is necessary to introduce a notion of "closeness" between facilities that is more relaxed than the definition used for covering, where we required two facilities to be at least $c_1 \cdot 5^r$ apart. Now we guarantee that if a facility is open then no other facility within distance $c_Y \cdot 5^r$ is opened, resulting in the following definition of $\mathrm{Y}(j, r)$ for every $\langle j, r\rangle \in \Pi$:

$$\mathrm{Y}(j, r) = \bigcup \{A(j', r) \mid \mathrm{d}(j, j') \leq c_Y \cdot 5^r\}, \qquad \text{where } c_Y = 2c_X + 3c_2.$$

The constant c_Y is chosen so that we can make sure that there are never two pairs $\langle j, r\rangle$ and $\langle j', r'\rangle$ such that both j and j' are open and $\mathrm{X}(j, r)$ and $\mathrm{X}(j', r')$ intersect. Thus, a client can always only belong to at most one set $\mathrm{X}(j, r)$, where $\langle j, r\rangle$ is open. The facility j will be the facility that the client is assigned to. The following lemmata follow as before:

▶ **Lemma 11.** *For any $\langle j, r\rangle \in \Pi$, $\mathrm{Y}(j, r)$ is a union of at most $2^{5\kappa}$ areas.*

▶ **Lemma 12.** *It holds that $\mathrm{Y}(j, r) \subseteq B(j, c_4 \cdot 5^r)$.*

▶ **Lemma 13.** *If $\langle j', r'\rangle$ is an ancestor of $\langle j, r\rangle$ in $\mathcal{T}$, then $\mathrm{Y}(j, r) \subseteq \mathrm{Y}(j', r')$.*

Coloring of pairs. We are now ready to define a tie-breaking rule based on colors. For any $\langle j, r\rangle \in \Pi$, consider the set $\Pi(j, r) = J_r \cap B(j, c_4 \cdot 5^r)$. By Lemma 3 and since $c_4 < 16c_1$, it follows that $\Pi(j, r)$ contains at most $2^{5\kappa}$ pairs. We need to guarantee that at most one of them will ever be opened. Thus we introduce a tie-breaking rule based on colors of pairs. This guarantees that out of all pairs in $\Pi(j, r)$ that fulfill the abundance condition only the one with the "smallest" color is opened.

More formally, we perform a preprocessing step using a greedy approach to color pairs of same log-radius in Π with $2^{5\kappa} + 1$ colors from 0 to $2^{5\kappa}$.

Specifically for each log-radius $r \in [\rho_{\min}, \rho_{\max}]$, we greedily color every pair $\langle j, r\rangle$ of J_r by one color s so that *no two pairs* $\langle j, r\rangle, \langle j', r\rangle \in J_r$ with $\mathrm{d}(j, j') \le c_4 \cdot 5^r$ *are colored with the same color*, and we refer to $\langle j, r, s\rangle$ as *triplet*. Let $J_{r,s} := \{\langle j, r, s'\rangle \in J_r \mid s' = s\}$.

Designated facilities. Furthermore, even if a triplet $\langle j, r, s\rangle$ fulfills the condition to be opened (which is explained in the next section) it will not be opened, if there is a "cheaper" facility nearby. More formally, we precompute for each pair $\langle j, r\rangle$ in Π the following *designated facility*: Let $f^*_{\langle j,r\rangle}$ be the minimum opening cost of any facility in $\mathrm{X}(j, r)$, i.e.,

$$f^*_{\langle j,r\rangle} = \min\{f_{j'} \mid j' \in J \cap \mathrm{X}(j, r)\}.$$

The *designated facility* $j^*_{\langle j,r\rangle}$ of $\langle j, r\rangle$ is the facility with minimum cost $f^*_{\langle j,r\rangle}$ in $\mathrm{X}(j, r)$ with ties broken according to the minimum id-number, i.e., $j^*_{\langle j,r\rangle} = \min\{j' \mid j' \in J \cap \mathrm{X}(j, r), f_{j'} = f^*_{\langle j,r\rangle}\}$.

▶ **Observation 14.** *For any $\langle j, r\rangle \in \Pi$, $\mathrm{d}(j, j^*_{\langle j,r\rangle}) \le c_3 \cdot 5^r$ and $f^*_{\langle j,r\rangle} \le f_j$.*

If the triplet $\langle j, r, s\rangle$ fulfills the condition to be opened, we open $j^*_{\langle j,r\rangle}$ instead, or do nothing if $j^*_{\langle j,r\rangle}$ is already open. Whenever all triplets for which a facility is designated are closed, then the facility is *closed*.

3 Processing updates

After the preprocessing phase we are given a laminar family of triplets, where each triplet $\langle j, r, s\rangle$ is formed by an area $A(j, r)$ along with its pre-defined color s. Depending on the set of clients $\mathcal{C}$, a triplet can be either *disabled* or *enabled* and either *open* or *closed*, where each open triplet is also enabled. These properties of triplets are maintained dynamically as the set $\mathcal{C}$ of clients changes. Initially $\mathcal{C} = \emptyset$ and all triplets are closed and disabled. We now proceed to the formal definitions.

Open triplets. We *open* a triplet $\langle j,r,s\rangle$ if there are enough clients in the set $X(j,r)$ to pay the opening cost and it has no strictly smaller-radius or no same-radius and strictly smaller-color open triplet in its "neighborhood". A triplet that is not open is *closed*. Formally a triplet $\langle j,r,s\rangle$ is open if it belongs to the set $J^{\text{open}}_{r,s}(\mathcal{C})$, which is defined[3] recursively as follows.

$$J^{\text{open}}_{r,s}(\mathcal{C}) = \Big\{ \langle j,r,s\rangle \in J_{r,s} \;\Big|\; 5^r \cdot |\mathcal{C} \cap \mathrm{X}(j,r)| \geq f^*_{\langle j,r\rangle} \wedge \forall \langle r',s'\rangle <_{\text{lex}} \langle r,s\rangle : \mathrm{Y}(j,r) \cap J^{\text{open}}_{r',s'}(\mathcal{C}) = \varnothing \Big\}.$$

We use $J^{\text{open}}_{\mathcal{C}} = \bigcup_{r,s} J^{\text{open}}_{r,s}(\mathcal{C})$ to denote the set of all open clients and $\mathcal{I}_{\mathcal{C}}$ to denote the set of all open facilities, i.e., $\mathcal{I}_{\mathcal{C}} = \{j^*_{\langle j,r\rangle} \in J \mid \langle j,r,s\rangle \in J^{\text{open}}_{\mathcal{C}}\}$.

We call the following condition for $\langle j,r,s\rangle$, used in the definition of $J^{\text{open}}_{r,s}$, the *abundance condition*,

$$5^r \cdot |\mathcal{C} \cap \mathrm{X}(j,r)| \geq f^*_{\langle j,r\rangle}. \tag{1}$$

▶ **Lemma 15.** *If $|\mathcal{C}| > 0$ then there exists at least one open triplet.*

When showing the bound on the approximation ratio, we need the property that for each point $i \in V$ there is *at most one* set $\mathrm{X}(j,r)$ associated with an open triplet such that i belongs to. This is necessary to make sure that each client "pays" for at most one open facility.

▶ **Lemma 16.** *Each client $i \in \mathcal{C}$ belongs to at most one $\mathrm{X}(j,r)$ with $\langle j,r,s\rangle \in J^{\text{open}}_{\mathcal{C}}$ for some color s.*

Note, however, that is not true that each client $i \in \mathcal{C}$ is contained in *at least one* $\mathrm{X}(j,r)$ associated with an open triplet for some r: even though there always exists a $\mathrm{X}(j,r)$ fulfilling the abundance condition and containing i (namely $\mathrm{X}(j^{\text{root}},\rho_{\max})$), the corresponding triplet $\langle j^{\text{root}},\rho_{\max},s\rangle$ might not be open due to a "nearby" open triplet of smaller logradius. To deal with this issue we introduce enabled triplets and show that each client in $\mathcal{C}$ is contained in at least one $\mathrm{X}(j,r)$ of an enabled triplet.

Enabled triplets. A triplet $\langle j,r,s\rangle$ is *enabled* if it belongs to the set $J^{\text{enabled}}_{r,s}(\mathcal{C})$, which is defined[4] as follows:

$$J^{\text{enabled}}_{r,s}(\mathcal{C}) = \Big\{ \langle j,r,s\rangle \in J_{r,s} \;\Big|\; \exists \langle r',s'\rangle \leq_{\text{lex}} \langle r,s\rangle : \mathrm{Y}(j,r) \cap J^{\text{open}}_{r',s'}(\mathcal{C}) \neq \varnothing \Big\}.$$

We use $J^{\text{enabled}}_{\mathcal{C}} = \bigcup_{r,s} J^{\text{enabled}}_{r,s}(\mathcal{C})$ to denote the set of all enabled facilities. The following observation follows from the definition.

▶ **Observation 17.** *If a triplet is open, then it is also enabled.*

Furthermore, as a corollary of Lemma 13 we have the following lemma.

▶ **Lemma 18.** *If a triplet is enabled, then all its ancestors in $\mathcal{T}$ are also enabled.*

[3] Remark that to make the formula a bit simpler we slightly abuse notation here – $\mathrm{Y}(j,r)$ is a set of areas (i.e., subsets of the metric space), while $J^{\text{open}}_{r',s'}(\mathcal{C})$ is a set of triples. Formally the intersection should be understood as $\mathrm{Y}(j,r) \cap \{j' \in J \mid \langle j',r',s'\rangle \in J^{\text{open}}_{r',s'}(\mathcal{C})\}$.

[4] Similarly to the definition of $J^{\text{open}}_{r,s}(\mathcal{C})$ we mean here $\mathrm{Y}(j,r) \cap \{j' \in J \mid \langle j',r',s'\rangle \in J^{\text{open}}_{r',s'}(\mathcal{C})\}$.

Since $A(j^{\text{root}}, \rho_{\max}) = V$, every point in V belongs to at least one $\mathrm{X}(j,r)$ associated with an enabled triplet. To guarantee that at least one triplet is enabled, recall that Lemma 15 showed that if $|\mathcal{C}| > 0$, then there exist an open, and, thus, also enabled triplet (see Observation 17). Lemma 18 implies that the root of $\mathcal{T}$ is enabled, implying the following lemma.

▶ **Lemma 19.** *If $|\mathcal{C}| > 0$ then every point in V belongs to at least one $\mathrm{X}(j,r)$ associated with an enabled triplet.*

The next lemma shows that the definition of enabled implies that any triplet that satisfies the abundance definition is either open or enabled. This is a crucial observation for the proof of the approximation ratio, as it will allow us to argue that for any facility that the optimal solution opens, an enabled triplet must be nearby.

▶ **Lemma 20.** *If $\langle j, r, s\rangle$ satisfies the abundance condition, then it is enabled.*

Assignment of clients. We next describe how to assign each client i to an enabled triplet $\langle j, r, s\rangle$ and an open facility. If $\langle j, r, s\rangle$ is not open, we show how to find a close open triplet of smallest radius. For this open triplet we know its designed facility that is open. This is the facility that the client is finally assigned to.

We start with the assignment of $i \in C$ to an enabled triplet. To this end, let r_i^{area} be the minimum logradius of any enabled triplet such that i belongs to the area associated with the triplet, i.e., $r_i^{\text{area}} = \min\{r \mid \langle j, r, s\rangle \in J_{\mathcal{C}}^{\text{enabled}}, i \in A(j,r)\}$. Note that $A(j,r) \subseteq \mathrm{X}(j,r)$, and let j_i^{area} be the center of the area with log-radius r_i^{area} and define $\langle j_i^{\text{area}}, r_i^{\text{area}}, s_i^{\text{area}}\rangle$ to be the corresponding triplet (recall that same-logradius areas are disjoint).

Once we determined the enabled triplet $\langle j_i^{\text{area}}, r_i^{\text{area}}, s_i^{\text{area}}\rangle$ of i, we assign i to the open triplet of minimum radius such that the corresponding facility belongs to $\mathrm{Y}(j_i^{\text{area}}, r_i^{\text{area}})$ and let j_i^{open} be the designated open facility of that open triplet. We assign i to it. Formally:

$$\begin{aligned}\langle r_i^{\text{aux}}, s_i^{\text{aux}}, j_i^{\text{aux}}\rangle &= \min\Big\{\langle r', s', j'\rangle \;\Big|\; \langle j', r', s'\rangle \in J_{\mathcal{C}}^{\text{open}}, \langle r', s'\rangle \le_{\text{lex}} \langle r_i^{\text{area}}, s_i^{\text{area}}\rangle,\\ &\qquad j' \in \mathrm{Y}(j_i^{\text{area}}, r_i^{\text{area}})\Big\},\\ j_i^{\text{open}} &= j^*_{\langle j_i^{\text{aux}}, r_i^{\text{aux}}\rangle}.\end{aligned}$$

Finally we denote the set of all clients assigned to facility j by $\mathcal{C}_j = \{i \in \mathcal{C} \mid j = j_i^{\text{open}}\}$. Note that j_i^{open} does not have to be the closest open facility, but as the next lemma shows it is not far away from an open facility.

▶ **Observation 21.** *Any $i \in \mathcal{C}$ is within $(c_2 + c_3 + c_4) \cdot 5^{r_i^{\text{area}}}$ of an open facility.*

The value r_i^{area} is crucial for the cost estimate. Thus it is important to characterize this value even further, as we do in the following lemma.

▶ **Lemma 22.** *For any $i \in \mathcal{C}$, if $i \in \mathrm{X}(j,r)$ and $\langle j, r, s\rangle \in J_{\mathcal{C}}^{\text{open}}$ for some color s, then $r_i^{\text{area}} = r$.*

Assume we open each facility in $\mathcal{I}_\mathcal{C}$ and each client is assigned to an open facility as described above or an even closer one, if one exists. As a consequence of the above lemma and the definition of open facilities we can now bound the total cost of the solution by $O(\sum_{i\in\mathcal{C}} 5^{r_i^{\text{area}}})$.

▶ **Lemma 23.** *It holds that $\sum_{i\in\mathcal{C}} \mathrm{d}(i, j_i) + \sum_{j\in\mathcal{I}_\mathcal{C}} f_j \le \sum_{i\in\mathcal{C}} (c_2 + c_3 + c_4 + 1) \cdot 5^{r_i^{\text{area}}}$.*

Now we are ready to prove the bound on the approximation ratio.

▶ **Theorem 24.** *For any subset $\mathcal{C} \subseteq V$ of clients, assign each client $i \in \mathcal{C}$ to the facility j_i^{open}. Then the cost of this solution is $O(1) \cdot \text{OPT}$, where OPT is the optimal solution for $\mathcal{C}$.*

Proof. Denote by $\mathcal{I}^*$ an arbitrary optimal solution. For each $i \in \mathcal{C}$ define j_i^* to be the facility i is connected to. Moreover, for each $j \in J$ let $\mathcal{C}_j^*$ be the set of clients connected to j. Formally, $j_i^* = \min\{j \in \mathcal{I}^* \,|\, \mathrm{d}(i,j) = \mathrm{d}(i, \mathcal{I}^*)\}$, $\mathcal{C}_j^* = \{i \in \mathcal{C} \,|\, j = j_i^*\}$. Consider some $j \in \mathcal{I}^*$ and let $r \in [\rho_{\min}, \rho_{\max}]$ be the logradius such that

$$5^{r-1} \cdot |\mathcal{C}_j^* \cap B(j, 5^{r-1})| < f_j \leq 5^r \cdot |\mathcal{C}_j^* \cap B(j, 5^r)|. \tag{2}$$

Note that r is well-defined as $5^{\rho_{\max}} \cdot |\mathcal{C}_j^* \cap B(j, 5^{\rho_{\max}})| \geq f_{\max} \geq f_j$ by the definition of $\rho_{\max}$ and $f_j \geq f_{\min} \geq 5^{\rho_{\min} + \log_5 n - 1} \geq 5^{\rho_{\min}-1} \cdot |\mathcal{C}_j^* \cap B(j, 5^{\rho_{\min}-1})|$ by the definition of $\rho_{\min}$.

To complete the proof we split $\mathcal{C}_j^*$ into two sets, $\mathcal{C}_j^{\text{lo}}$ and $\mathcal{C}_j^{\text{hi}}$, according to whether $i \in B(j, 5^{r-1})$ or not, i.e., $\mathrm{d}(i,j) \leq 5^{r-1}$ or $\mathrm{d}(i,j) > 5^{r-1}$ respectively, and show that

$$\sum_{i \in \mathcal{C}_j^{\text{lo}}} 5^{r_i^{\text{area}}} < 5 \cdot f_j, \text{ and} \tag{A}$$

$$\sum_{i \in \mathcal{C}_j^{\text{hi}}} 5^{r_i^{\text{area}}} \leq \sum_{i \in \mathcal{C}_j^{\text{hi}}} 5 \cdot \mathrm{d}(i,j). \tag{B}$$

Before showing the above inequalities we first argue that they prove the bound on the approximation ratio. Note that every client belongs to $\mathcal{C}_j^*$ for some $j \in \mathcal{I}^*$. Thus,

$$\sum_{i \in \mathcal{C}} 5^{r_i^{\text{area}}} \leq \sum_{j \in \mathcal{I}^*} 5 \cdot f_j + \sum_{i \in \mathcal{C}} 5 \cdot \mathrm{d}(i,j) \leq 5 \cdot \text{OPT}.$$

Using Lemma 23 we get that

$$\sum_{i \in \mathcal{C}} \mathrm{d}(i, j_i^{\text{open}}) + \sum_{j \in \mathcal{I}_\mathcal{C}} f_j \leq \sum_{i \in \mathcal{C}} (c_2 + c_3 + c_4 + 1) \cdot 5^{r_i^{\text{area}}} \leq 5(c_2 + c_3 + c_4 + 1) \cdot \text{OPT}.$$

We first show (A). Consider $i \in \mathcal{C}_j^{\text{lo}}$, or equivalently, $i \in B(j, 5^{r-1})$. We claim that $r_i^{\text{area}} \leq r$. Indeed, if $\langle j, r, s \rangle \in \Pi$ for some color s, then $\mathrm{X}(j,r) \supseteq A(j,r) \supseteq B(j, 5^r)$ along with (2) give

$$5^r \cdot |\mathcal{C} \cap \mathrm{X}(j,r)| \geq 5^r \cdot |\mathcal{C}_j^* \cap B(j, 5^r)| \geq f_j \geq f^*_{\langle j, r \rangle},$$

which in turn implies that $\langle j, r, s \rangle$ satisfies the abundance condition and so it is enabled (Observation 20). By definition of r_i^{area}, we get $r_i^{\text{area}} \leq r$. If $\langle j, r, s \rangle \notin \Pi$ for any s, then by Lemma 6, there exists a triplet $\langle j', r, s \rangle$ such that $i \in A(j', r)$. Because $\mathrm{d}(j', j) \leq \mathrm{d}(j', i) + \mathrm{d}(i, j) \leq (c_2 + 1) \cdot 5^r$ we have that $B(j, 5^r) \subseteq \mathrm{X}(j', r)$. This along with (2) imply that

$$5^r \cdot |\mathcal{C} \cap \mathrm{X}(j', r)| \geq 5^r \cdot |\mathcal{C}_j^* \cap B(j, 5^r)| \geq f_j \geq f^*_{\langle j', r \rangle},$$

where the last inequality follows by definition of $f^*_{\langle j', r \rangle}$. Similarly it follows that $\langle j', r, s \rangle$ is enabled and $r_i^{\text{area}} \leq r$. Recalling that $i \in B(j, 5^{r-1})$ we finally arrive at

$$\sum_{i \in \mathcal{C}_j^{\text{lo}}} 5^{r_i^{\text{area}}} \leq \sum_{i \in \mathcal{C}_j^{\text{lo}}} 5^r \leq 5 \cdot 5^{r-1} \cdot |\mathcal{C}_j^* \cap B(j, 5^{r-1})| \leq 5 \cdot f_j.$$

We next show (B). First, observe that for any ball $B(j, 5^{r''})$, $r'' \geq r$ with $i \in B(j, 5^{r''})$, one can prove similarly to the above that there exists an *enabled* triplet $\langle j'', r'', s \rangle$ such that

$i \in A(j'', r'')$. We have that $\mathrm{d}(j'', j) \le \mathrm{d}(j'', i) + \mathrm{d}(i, j) \le (c_2 + 1) \cdot 5^{r''}$, thus $X(j'', r'') \supseteq B(j, 5^{r''}) \supseteq B(j, 5^r)$. This, together with (2) and $f^*_{\langle j'',r''\rangle} \le f_j$ give the claim.

Now, consider $i \in \mathcal{C}_j^{\text{hi}}$. Since $\mathrm{d}(i, j) > 5^{r-1}$, we get that $\lceil \log_5 \mathrm{d}(i, j)\rceil \ge r$ and $i \in B(j, 5^{\lceil \log_5 \mathrm{d}(i,j)\rceil})$. By the discussion above, there exists an enabled triplet $\langle j'', \lceil \log_5 \mathrm{d}(i, j)\rceil, s\rangle$ such that $\mathrm{X}(j'', \lceil \log_5 \mathrm{d}(i, j)\rceil) \supseteq B(j, 5^{\lceil \log_5 \mathrm{d}(i,j)\rceil})$, implying that $r_i^{\text{area}} \le \lceil \log_5 \mathrm{d}(i, j)\rceil$ and so

$$\sum_{i \in \mathcal{C}_j^{\text{hi}}} 5^{r_i^{\text{area}}} \le \sum_{i \in \mathcal{C}_j^{\text{hi}}} 5 \cdot \mathrm{d}(i, j). \qquad ◀$$

4 Data Structure

In this section we devise a data structure for the dynamic metric facility location problem that supports insertions and deletions of clients as well as returning (a) the approximate cost of the optimal solution or (b) a set of open facilities that achieves this approximate cost. We achieve this by maintaining the minimum cost solution restricted to pairs in Π. By Theorem 24 this is a $O(1)$ approximation to the cost of the optimal solution.

From the preprocessing phase the algorithm is given the set Π of facility-radius-color triplets, as well as the laminar family of areas $\mathcal{A}$ with its dependency tree $\mathcal{T}$ using the following representation. (1) A two-dimensional array of size $(\rho_{\max} - \rho_{\min} + 1) \times (2^{5\kappa} + 1)$, keeping for each logradius $r \in [\rho_{\min}, \rho_{\max}]$ and color s a list of all the facilities of J_r that share the color s, and (2) the dependency tree $\mathcal{T}$ in a tree data structure. Whenever we use the term *subtree*, *child*, or *descendant* in the following we refer to the dependency tree. (3) For each triplet $v = \langle j, r, s\rangle \in \Pi$, the list `neighbors_above`$(v)$ of all triplets $\langle j', r', s'\rangle$ such that (a) $\langle r', s'\rangle >_{\text{lex}} \langle r, s\rangle$ and (b) $j \in \mathrm{Y}(j', r')$. (4) For each triplet $v = \langle j, r, s\rangle \in \Pi$, the value $f^*_{\langle j,r\rangle}$, which is the minimum opening cost among all facilities in $\mathrm{X}(j, r)$. Using the algorithm in Subsection 4.1 each list `neighbors_above` and each value $f^*_{\langle j,r\rangle}$ can be computed in time $O(2^{O(\kappa)}\Delta)$. Thus, the above data structure can be built in time $O(|J| \cdot 2^{O(\kappa)}\Delta)$.

The algorithm will maintain a dynamic data structure, which can be viewed as an *annotated dependency tree* that keeps for each node $v = \langle j, r, s\rangle$ of $\mathcal{T}$ the following information:

1. three bits $\mathrm{O_x}(v)$, $\mathrm{E_x}(v)$, $\mathrm{A_x}(v)$, which indicate whether the triplet $\langle j, r, s\rangle$ is open, enabled and fulfils the abundance condition, respectively,
2. the number $n_{\text{area}}(v)$ of current clients that belong to the area $A(j, r)$, i.e., $n_{\text{area}}(v) = |\mathcal{C} \cap A(j, r)|$,
3. the number $n_{\mathrm{x}}(v)$ of current clients that belong to $\mathrm{X}(j, r)$, i.e., $n_{\mathrm{x}}(v) = |\mathcal{C} \cap \mathrm{X}(j, r)|$,
4. the number `openbelow`(v) of all open triplets $\langle j', r', s'\rangle$ with $\langle r', s'\rangle <_{\text{lex}} \langle r, s\rangle$ and their corresponding facilities falling within $\mathrm{Y}(j, r)$, i.e.,
$$\texttt{openbelow}(v) = \left|\left\{\langle j', r', s'\rangle \in J_{\mathcal{C}}^{\text{open}} \mid \langle r', s'\rangle <_{\text{lex}} \langle r, s\rangle, j' \in \mathrm{Y}(j, r)\right\}\right|,$$
5. the number $n_{\text{enblbelow}}(v)$ of current clients that belong to areas below that are enabled, i.e.,
$$n_{\text{enblbelow}}(v) = \left|\mathcal{C} \cap \bigcup\left\{\langle j', r', s'\rangle \in J_{\mathcal{C}}^{\text{enbl}} \mid A(j', r') \subset A(j, r)\right\}\right|,$$
6. the value $\mathrm{c}(v) = \sum\{5^{r_i^{\text{area}}} \mid i \in \mathcal{C} \cap A(j, r)\}$ (note that with the currently open facilities the cost accrued for the clients in $A(j, r)$ is $O(\mathrm{c}(v))$),
7. the value $y(v)$, which is the cost of the children of v, i.e., $y(v) = \sum_{u \text{ child of } v} \mathrm{c}(u)$.

We next describe the usefulness of the information we keep. Points 1-2 are self-explanatory. Point 3 provides information to test the abundance condition, and thus update the bits in Point 1. Point 4 is useful when deciding whether we should open an area or not. Points 5-7 allow us to efficiently update the cost of the solution.

4.1 Finding all balls containing a given point

In this section we describe a crucial subroutine that we use repeatedly when handling updates. It is given the hierarchy data structure for $\mathcal{T}$, an arbitrary point $p \in V$ and some constant c^* such that $c^* \geq (5/4)c_1$ and returns all balls $\langle j, r \rangle \in \Pi$ that are at distance at most $c^* \cdot 5^r$ from p, i.e., $p \in B(j, c^* \cdot 5^r)$. For $r \in [\rho_{\max}, \rho_{\min}]$, let $S(r)$ denote the set of such balls.

The algorithm FindBalls(p, c^*) performs a top-down traversal of the tree starting at its root $\langle j, \rho_{\max} \rangle$. Note that by the definition of $\rho_{\max}$, all points belong to $B(j, c^* \cdot 5^{\rho_{\max}})$ and $S(\rho_{\max}) = \{\langle j, \rho_{\max} \rangle\}$. For computing $S(r)$, $r = (\rho_{\max} - 1)$, it determines all children of the root to find the pairs $\langle j', r \rangle$ such that the distance of j' and p is at most $c^* \cdot 5^r$. This step is repeated to compute the set $S(\ell)$ for every level of the hierarchy, until we reach the bottom-most level. Finally, we let $S \coloneqq \bigcup \{S(r) : r \in [\rho_{\min}, \rho_{\max}]\}$. A detailed description of this procedure is deferred to the full version.

We next show that the algorithm correctly computes the set $S(r)$, for every log-radius r. Define $\text{children}(S(r)) = \bigcup_{\langle j,r \rangle \in S(r)} \text{children}(j, r)$.

▶ **Lemma 25.** *For each logradius $r \in [\rho_{\max}, \rho_{\min})$ assume $S(r)$ is computed correctly. Then it holds that $S(r-1) \subseteq \text{children}(S(r))$.*

Proof. Assume towards contradiction that there exists $\langle j, r-1 \rangle \in S(r-1)$ such that $\mathrm{d}(j, p) \leq c^* \cdot 5^{r-1}$ but $\langle j, r-1 \rangle \notin \text{children}(S(r))$. Let $\langle j', r \rangle$ be the parent of $\langle j, r-1 \rangle$ in $\mathcal{T}$. By Lemma 2, $\mathrm{d}(j, j') \leq c_1 \cdot 5^r$.

Now, since $S(r)$ is correct, it follows that $\langle j', r \rangle \notin S(r)$, and thus $\mathrm{d}(p, j') > c^* \cdot 5^r$. However, by Lemma 2 we get that $p \in B(j, c^* \cdot 5^{r-1}) \subseteq B(j', c^* \cdot 5^r)$ and thus $\mathrm{d}(p, j') \leq c^* \cdot 5^r$, which is a contradiction. Thus the lemma follows. ◀

Since $c_X \geq (5/4)c_1$, let $c^* = c_X$. We now argue about the running time of FindBalls(p, c^*). Note that for each logradius r, if $p \in B(j, c^* \cdot 5^r)$, then $\mathrm{d}(p, j) \leq c^* \cdot 5^r$. By Lemma 3 and the fact that $c^* \leq 4c_1$ it follows that $|S(r)| \leq 2^{3\kappa}$. Additionally, by Lemma 4 each pair in $S(r)$ has at most $2^{4\kappa}$ children in $\mathcal{T}$ and there are at most Δ different radii. Thus the running time of the algorithm and the size of the output set S are both bounded by $2^{7\kappa} \cdot \Delta$.

▶ **Lemma 26.** *The running time of* FindBalls(p, c^*) *and the size of the output set S are both bounded by $2^{7\kappa} \cdot \Delta$.*

Repeatedly applying the FindBalls subroutine and updating the tree hierarchy in a bottom-up fashion, we can show that insertions and deletions of clients can be handled in $O(2^{O(\kappa^2)} \cdot \Delta^3 \cdot (\kappa^2 + \log \Delta))$ time. Additionally note that under client updates, the value of n will change, which in turn causes $\rho_{\min}$ to either increase or decrease by one. This forces us to either add or delete a bottom-level in the hierarchy, which can be implemented in $O((|J| + |\mathcal{C}|) \cdot 2^{O(\kappa^2)} \cdot \Delta^3 \cdot (\kappa^2 + \log \Delta))$ time. Since such an update is required only after $\Theta(n)$ operations, we get that the amortized time of our algorithm is still bounded by $O(2^{O(\kappa^2)} \cdot \Delta^3 \cdot (\kappa^2 + \log \Delta))$. By employing a standard global rebuilding technique we achieve a worst-case update time, thus proving our main result in Theorem 1. Details on implementing the above steps are deferred to the full version.

References

1 Sebastian Abshoff, Peter Kling, Christine Markarian, Friedhelm Meyer auf der Heide, and Peter Pietrzyk. Towards the price of leasing online. *Journal of Combinatorial Optimization*, 4(32):1197–1216, 2015.

2 Hyung-Chan An, Ashkan Norouzi-Fard, and Ola Svensson. Dynamic facility location via exponential clocks. In *Symposium on Discrete Algorithms (SODA)*, pages 708–721, 2015.

3 Aris Anagnostopoulos, Russell Bent, Eli Upfal, and Pascal Van Hentenryck. A simple and deterministic competitive algorithm for online facility location. *Information and Computation*, 194(2):175–202, 2004.

4 Barbara Anthony and Anupam Gupta. Infrastructure leasing problems. *International Conference on Integer Programming and Combinatorial Optimization*, pages 424–438, 2007.

5 Pierre Chardaire, Alain Sutter, and Marie-Christine Costa. Solving the dynamic facility location problem. *Networks*, 28(2):117–124, 1996.

6 Artur Czumaj, Christiane Lammersen, Morteza Monemizadeh, and Christian Sohler. $(1+\epsilon)$-approximation for facility location in data streams. In *Symposium on Discrete Algorithms (SODA)*, pages 1710–1728, 2013.

7 Zvi Drezner and Horst W Hamacher. *Facility location: applications and theory*. Springer Science & Business Media, 2001.

8 David Eisenstat, Claire Mathieu, and Nicolas Schabanel. Facility location in evolving metrics. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 459–470, 2014.

9 Reza Zanjirani Farahani, Maryam Abedian, and Sara Sharahi. Dynamic facility location problem. In *Facility Location*, pages 347–372. Springer, 2009.

10 Dimitris Fotakis. A primal-dual algorithm for online non-uniform facility location. *Journal of Discrete Algorithms*, 5(1):141–148, 2007.

11 Dimitris Fotakis. On the competitive ratio for online facility location. *Algorithmica*, 50(1):1–57, 2008.

12 Dimitris Fotakis. Memoryless facility location in one pass. *ACM Trans. Algorithms*, 7(4):49:1–49:24, 2011.

13 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31(1):228–248, 1999.

14 Monika Henzinger, Dariusz Leniowski, and Claire Mathieu. Dynamic clustering to minimize the sum of radii. In *European Symposium on Algorithms (ESA)*, 2017.

15 Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *Symposium on Theory of Computing (STOC)*, pages 373–380, 2004.

16 Christiane Lammersen and Christian Sohler. Facility location in dynamic geometric data streams. In *European Symposium on Algorithms (ESA)*, pages 660–671, 2008.

17 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45–58, 2013.

18 Adam Meyerson. Online facility location. In *Symposim on Foundations of Computer Science (FOCS)*, pages 426–431, 2001.

19 Chandrashekhar Nagarajan and David P Williamson. Offline and online facility leasing. *Discrete Optimization*, 10(4):361–370, 2013.

20 Maxim Sviridenko. An improved approximation algorithm for the metric uncapacitated facility location problem. In *International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 240–257, 2002.

21 George O Wesolowsky. Dynamic facility location. *Management Science*, 19(11):1241–1248, 1973.

Dynamic Effective Resistances and Approximate Schur Complement on Separable Graphs

Gramoz Goranci
University of Vienna, Faculty of Computer Science, Vienna, Austria
gramoz.goranci@univie.ac.at

Monika Henzinger
University of Vienna, Faculty of Computer Science, Vienna, Austria
monika.henzinger@univie.ac.at

Pan Peng[1]
Department of Computer Science, University of Sheffield, Sheffield, UK
p.peng@sheffield.ac.uk

Abstract

We consider the problem of dynamically maintaining (approximate) all-pairs effective resistances in separable graphs, which are those that admit an n^c-separator theorem for some $c < 1$. We give a fully dynamic algorithm that maintains $(1+\varepsilon)$-approximations of the all-pairs effective resistances of an n-vertex graph G undergoing edge insertions and deletions with $\tilde{O}(\sqrt{n}/\varepsilon^2)$ worst-case update time and $\tilde{O}(\sqrt{n}/\varepsilon^2)$ worst-case query time, if G is guaranteed to be $\sqrt{n}$-separable (i.e., it is taken from a class satisfying a $\sqrt{n}$-separator theorem) and its separator can be computed in $\tilde{O}(n)$ time. Our algorithm is built upon a dynamic algorithm for maintaining *approximate Schur complement* that approximately preserves pairwise effective resistances among a set of terminals for separable graphs, which might be of independent interest.

We complement our result by proving that for any two fixed vertices s and t, no incremental or decremental algorithm can maintain the $s - t$ effective resistance for $\sqrt{n}$-separable graphs with worst-case update time $O(n^{1/2-\delta})$ and query time $O(n^{1-\delta})$ for any $\delta > 0$, unless the Online Matrix Vector Multiplication (OMv) conjecture is false.

We further show that for *general* graphs, no incremental or decremental algorithm can maintain the $s - t$ effective resistance problem with worst-case update time $O(n^{1-\delta})$ and query-time $O(n^{2-\delta})$ for any $\delta > 0$, unless the OMv conjecture is false.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Dynamic graph algorithms, effective resistance, separable graphs, Schur complement, conditional lower bounds

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.40

Related Version A full version of the paper is available at [17], `https://arxiv.org/abs/1802.09111`.

Funding The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506.

[1] Work done in part while at the Faculty of Computer Science, University of Vienna, Austria

© Gramoz Goranci, Monika Henzinger, and Pan Peng;
licensed under Creative Commons License CC-BY

1 Introduction

Effective resistances and the closely related electrical flows are basic concepts for resistor networks [12] and were found to be very useful in the design of graph algorithms, e.g., for computing and approximating maximum flow [8, 36, 37], random spanning tree generation [38, 43], multicommodity flow [27], oblivious routing [18], and graph sparsification [44, 11]. They also have found applications in social network analysis, e.g., for measuring the similarity of vertices in social networks [33], in machine learning, e.g., for Gaussian sampling [7] and in chemistry, e.g., for measuring chemical distances [28]. Previous research has studied the problem of how to quickly compute and approximate the effective resistances (or equivalently, *energies* of electrical flows; see the full version for more discussions), as such algorithms can be used as a crucial subroutine for other graph algorithms. For example, one can $(1+\varepsilon)$-approximate the $s-t$ effective resistance in $\tilde{O}(m+n\varepsilon^{-2})$ [14] and $\tilde{O}(m\log(1/\varepsilon))$ [9] time, respectively, in any n-vertex m-edge weighted graph, for any two vertices s,t. (Throughout the paper, we use $\tilde{O}$ to hide polylogarithmic factors, i.e., $\tilde{O}(f(n))=O(f(n)\cdot \operatorname{poly}\log f(n))$.) There are also algorithms that find $(1+\varepsilon)$-approximations to the effective resistance between every pair of vertices in $\tilde{O}(n^2/\varepsilon)$ time [24]. In order to exactly compute the $s-t$ (or single-pair) and all-pairs effective resistance(s), the current fastest algorithms run in times $O(n^\omega)$ (by using the fastest matrix inversion algorithm [6, 21]) and $O(n^{2+\omega})$, respectively, where $\omega < 2.373$ is the matrix multiplication exponent [46]. In planar graphs, the algorithms for exactly computing $s-t$ and all-pairs effective resistance(s) run in times $O(n^{\omega/2})$ (by the nested dissection method for solving linear system in planar graphs [34]) and $O(n^{2+\omega/2})$, respectively.

A natural algorithmic question is how to efficiently maintain the effective resistances *dynamically*, i.e., if the graph undergoes edge insertions and/or deletions, and the goal is to support the update operations and query for the effective resistances as quickly as possible, rather than having to recompute it from scratch each time. Besides the potential applications in the design of other (dynamic) algorithms, it is also of practical interest, e.g., to quickly report the (dis)similarity between any two nodes in a social network in which its members and their relationship are constantly changing. So far our understanding towards this question is very limited: for exact maintenance, the only approach (for single-pair effective resistance) we are aware of is to invoke the dynamic matrix inversion algorithm which gives $O(n^{1.575})$ update time and $O(n^{0.575})$ query time or $O(n^{1.495})$ update time and $O(n^{1.495})$ query time [42]; for $(1+\varepsilon)$-approximate maintenance, we can maintain the spectral sparsifier of size $n\operatorname{poly}(\log n,\varepsilon^{-1})$ with $\operatorname{poly}(\log n,\varepsilon^{-1})$ update time [3], while answering each query will cost $\Theta(n\operatorname{poly}(\log n,\varepsilon^{-1}))$ time. (Subsequent to the Arxiv submission [17] of this paper, Durfee et al. obtained a fully dynamic algorithm that maintains $(1+\varepsilon)$-approximations to all-pairs effective resistances of an unweighted, undirected multi-graph with $\tilde{O}(m^{4/5}\varepsilon^{-4})$ expected amortized update and query time [13].)

In this paper, we study the problem of dynamically maintaining the (approximate) effective resistances in *separable graphs*, which are those that satisfies an n^c-separator theorem for some $c<1$. Interesting classes of separable graphs include planar graphs, minor free graphs, bounded-genus graphs, almost planar graphs (e.g., road networks) [35], most 3-dimensional meshes [40] as well as many real-world networks (e.g., phone-call graphs, Web graphs, Internet router graphs) [5]. In the static setting, effective resistances (or electrical flows) in planar/separable graphs have been utilized by Miller and Peng [39] to obtain the first $\tilde{O}(\frac{m^{6/5}}{\varepsilon^{\Theta(1)}})$ time algorithm for approximate maximum flow in such graphs, and have also been studied by Anari and Oveis Gharan [4] in the analysis of an approximation algorithm for

Asymmetric TSP. We now give the necessary definitions to state our results.

Effective Resistances. Let $G = (V, E, \mathbf{w})$ be a undirected weighted graph with $\mathbf{w}(e) > 0$ for any $e \in E$. Let $\mathbf{A}$ denote its weighted adjacency matrix and $\mathbf{D}$ denote the weighted degree diagonal matrix. Let $\mathbf{L} = \mathbf{D} - \mathbf{A}$ denote the *Laplacian* matrix of G. Let $\mathbf{L}^\dagger$ denote the Moore-Penrose pseudo-inverse of the Laplacian of G. Let $\mathbf{1}_u \in \mathbb{R}^V$ denote the indicator vector of vertex u such that $\mathbf{1}_u(v) = 1$ if $v = u$ and 0 otherwise. Let $\chi_{s,t} = \mathbf{1}_s - \mathbf{1}_t$. Given any two vertices $u, v \in V$, the $s - t$ *effective resistance* is defined as $R_G(s,t) := \chi_{s,t}^T \mathbf{L}^\dagger \chi_{s,t}$.

Separable Graphs. Let $\mathcal{C}$ be a class of graphs that is closed under taking subgraphs. We say that $\mathcal{C}$ satisfies a $f(n)$*-separator theorem* if there are constants $\alpha < 1$ and $\beta > 0$ such that every graph in S with n vertices has a cut set with at most $\beta f(n)$ vertices that separates the graph into components with at most αn vertices each [35]. In this paper we are particularly interested in the class of graphs that satisfies an $n^{1/2}$-separator theorem, which include the class of planar graphs, K_t-minor free graphs and bounded-genus graphs, etc., though our approach can also be generalized to other class of graphs that satisfies a n^c-separator theorem, for some $c < 1$. In the following, we call a graph $f(n)$*-separable* if it is a member of a class that satisfies an $f(n)$-separator theorem.

We would like to quickly maintain the exact or a good approximation of the $s - t$ effective resistances in a $\sqrt{n}$-separable graph that undergoes edge insertions and deletions, for all pairs $s, t \in V$. We call this the *dynamic all-pairs effective resistances problem.* Our goal is to solve this problem with both small update and query times. More precisely, our data structure supports the following operations.

- Insert(u, v, w): Insert the edge (u, v) of weight w to G, provided that the updated graph remains $\sqrt{n}$-separable.
- Delete(u, v): Delete the edge (u, v) from G.
- EffectiveResistance(s, t): Return the exact or approximate value of the effective resistance between s and t in the current graph G.

1.1 Our Results

We give a fully dynamic algorithm for maintaining $(1 + \varepsilon)$-approximations of all-pairs and single-pair effective resistance(s) with small update and query times for any $\sqrt{n}$-separable graph, if its separator can be computed fast. Throughout the paper, all the running times of our algorithms are measured in *worst-case* performance. All our algorithms are randomized, and the performance guarantees hold with probability at least $1 - n^{-c}$ for some $c \geq 1$.

▶ **Theorem 1.** *Let G denote a dynamic n-vertex graph under edge insertions and deletions. Assume that G is $\sqrt{n}$-separable and its separator can be computed in $s(n)$ time, throughout the updates. There exist fully dynamic algorithms that maintain $(1 + \varepsilon)$-approximations of*

- *the all-pairs effective resistances with $\tilde{O}(\frac{\sqrt{n}}{\varepsilon^2} + \frac{s(n)}{\sqrt{n}})$ update time and $\tilde{O}(\frac{\sqrt{n}}{\varepsilon^2})$ query time;*
- *the $s - t$ effective resistance with $\tilde{O}(\frac{\sqrt{n}}{\varepsilon^2} + \frac{s(n)}{\sqrt{n}})$ update time and $O(1)$ query time.*

In particular, if $s(n) = \tilde{O}(n)$, then our update times are $\tilde{O}(\frac{\sqrt{n}}{\varepsilon^2})$.

By using the well known facts that a balanced separator of size $O(\sqrt{n})$ for planar graphs (and bounded-genus graphs) can be computed in $O(n)$ time [35], and for K_t-minor-free graphs (for any fixed integer $t > 0$) in $O(n^{1+\delta})$ time, for any constant $\delta > 0$ [26], we obtain dynamic algorithms for the effective resistances for planar and minor-free graphs with $\tilde{O}(\sqrt{n}/\varepsilon^2)$ and $\tilde{O}(\sqrt{n}/\varepsilon^2 + n^{1/2+\delta})$ update time, respectively.

The performance of our dynamic algorithm in planar graphs almost matches the best-known dynamic algorithm for $(1+\varepsilon)$-approximate all-pairs shortest path in planar graphs with $\tilde{O}(\sqrt{n})$ update and query time [2], though our approaches are different. This is interesting as the shortest path corresponds to flows with controlled ℓ_1 norm while the energy of electrical flows (i.e., effective resistance) corresponds to those with minimum ℓ_2 norm.

In order to design a dynamic algorithm for effective resistances of separable graphs (i.e., to prove Theorem 1), we give a fully dynamic algorithm that efficiently maintains an *approximate Schur complement* [30, 31, 14] of such graphs (see Section 4.1), which might be of independent interest. Approximate Schur complement can be treated as a *vertex sparsifier* that preserves pairwise effective resistances among a set of terminals (see Section 3). Therefore, our algorithm is a dynamic algorithm for *vertex effective resistance sparsifiers* with sublinear (in n) update time for separable graphs. The problem of dynamically maintaining graph *edge sparsifiers* has received attention very recently. For example, Abraham et al. presented fully dynamic algorithms that maintain cut and spectral sparsifiers with poly-logarithmic update times [3]. Formally, we prove the following theorem.

▶ **Theorem 2.** *For an n-vertex $\sqrt{n}$-separable graph G whose separator can be computed in $s(n)$ time, and a terminal set $K \subseteq V$ with $|K| \leq O(\sqrt{n})$, there exists a fully dynamic algorithm that maintains a $(1+\delta)$-approximate Schur complement with respect to K' such that $K \subseteq K'$ and $|K'| = O(\sqrt{n})$, while achieving $\tilde{O}(\sqrt{n}/\delta^2 + \frac{s(n)}{\sqrt{n}})$ update time. Furthermore, our algorithm supports terminal additions as long as $|K| \leq O(\sqrt{n})$.*

We complement our algorithm by giving a conditional lower bound for any *incremental* or *decremental* algorithm that maintains *single-pair* effective resistance of a $\sqrt{n}$-separable graph. Our lower bound is established from the *Online Matrix Vector Multiplication* (*OMv*) *conjecture* (see the full version).

▶ **Theorem 3.** *No incremental or decremental algorithm can maintain the (exact) $s-t$ effective resistance in $\sqrt{n}$-separable graphs on n vertices with both $O(n^{\frac{1}{2}-\delta})$ worst-case update time and $O(n^{1-\delta})$ worst-case query time for any $\delta > 0$, unless the OMv conjecture is false.*

We note that there are very few conditional lower bounds for dynamic *planar/separable* graphs, as most known reductions are highly non-planar. The only recent result that we are aware of is by Abboud and Dahlgaard [1], who showed that under some popular conjecture, no algorithm for dynamic shortest paths or maximum weight bipartite matching in planar graphs has both updates and queries in amortized $O(n^{1/2-\delta})$ time, for any $\delta > 0$.

We also give a stronger conditional lower bound for the same problem in *general* graphs, which shows that it is hard to maintain effective resistances with both sublinear (in n) update and query times for general graphs, even for the incremental or decremental setting.

▶ **Theorem 4.** *No incremental or decremental algorithm can maintain the (exact) $s-t$ effective resistance in general graphs on n vertices with both $O(n^{1-\delta})$ worst-case update time and $O(n^{2-\delta})$ worst-case query time for any $\delta > 0$, unless the OMv conjecture is false.*

We remark that both lower bounds for separable and general graphs hold for any algorithm with sufficiently high accurate approximation ratio (see Section 5 and full version).

Comparison to [16]. In our previous work [16], we gave a fully dynamic algorithm for $(1+\varepsilon)$-approximating all-pairs effective resistances for planar graphs with $\tilde{O}(r/\varepsilon^2)$ update time and $\tilde{O}((r+n/\sqrt{r})/\varepsilon^2)$ query time, for any r larger than some constant. The algorithm can also be generalized to $\sqrt{n}$-separable graphs, and we also provided a conditioned lower bound for any approximation algorithm of the $s-t$ effective resistance in general graphs in the *vertex-update* model. However, besides the apparent improvement of the performance of the dynamic algorithm (i.e., we reduce the best trade off between update time and query time from $\tilde{O}(n^{2/3})$ and $\tilde{O}(n^{2/3})$ to $\tilde{O}(n^{1/2})$ and $\tilde{O}(n^{1/2})$), our current work also improves over and differs from [16] in the following perspectives.

I. Our algorithm dynamically maintains the approximate Schur complement of a separable graphs by maintaining a separator tree of such graphs, rather than their *r-divisions* as used in [16]. In fact, we do not believe purely r-divisions based algorithms will achieve the performance as guaranteed by our new algorithm. This is evidenced by previous dynamic algorithms for maintaining reachability in directed planar graphs by Subramanian [45], $(1+\varepsilon)$-approximating to all-pairs shortest paths by Klein and Subramanian [29], exactly maintaining $s-t$ max-flow in planar graphs by Italiano et al. [23], all of which are based on r-divisions and have running times of order $n^{2/3}$ (and some of which have been improved by using other approaches).

II. Our current lower bound is much stronger than the previous one: the previous lower bound only holds for general graphs and the *vertex-update* model, where nodes, not edges, are turned on or off, and its proof was based on a simple relation between $s-t$ connectivity and $s-t$ effective resistance $R_G(s,t)$ (i.e., if s,t is connected iff $R_G(s,t)$ is not infinity). In contrast, our new lower bounds hold for separable graphs (and also general graphs) and the edge-update model. The corresponding proofs exploit new reductions from the OMv problem to the 5-length cycle detection and triangle detection problems in separable graphs and general graphs, respectively, which might be of independent interest, and the latter problems are related to the effective resistances (see Section 5).

1.2 Our Techniques

Our dynamic algorithm for maintaining an Approximate Schur complement (ASC) w.r.t. a set of terminals for separable graphs is built upon maintaining a *separator tree* of such graphs and two properties (called *transitivity* and *composability*) of ASCs. Such a tree can be constructed very efficiently by recursively partitioning the subgraphs using separators. Slightly more formally, each node in the tree corresponds to a subgraph of the original graph and contains a subset of vertices as its boundary vertices which in turn are treated as terminals. For each node H, we will maintain an ASC H' of H w.r.t its terminals. We will guarantee throughout all the updates that the ASC of any node can be computed efficiently in a bottom-up fashion, by the above two properties of ASCs. This stems from the fact that we only need to recompute the ASCs of nodes that lie on a path from a *constant* number leaves to the node of interest. Since each such path has length $O(\log n)$ and the recomputation of ASC of one node takes time $\tilde{O}(\sqrt{n})$, the update time will be guaranteed to be $\tilde{O}(\sqrt{n})$. For the detailed implementation, we need to overcome the difficulty that the error in the approximation ratio might accumulate through this recursive computation and an update might require to change the set of boundary vertices of many nodes, thus resulting in a prohibitive running time. We remark that though the idea of using separator tree of planar/separable graphs is standard (e.g., [15]), the main novelty of our algorithm is to use such a tree as the backbone to dynamically maintain the approximate Schur complement.

To obtain our dynamic algorithms for all-pair effective resistance, we appropriately declare and add new terminals whenever we get a new query, and then run the above dynamic algorithm for ASC with respect to the corresponding terminal set.

To obtain our lower bound, we provide new reductions from the Online Boolean Matrix-Vector Multiplication (OMv) problem to the incremental or decremental single-source effective resistance problem. More specifically, given an OMv instance with vectors $\mathbf{u}, \mathbf{v}$ and a matrix $\mathbf{M}$, we construct a $\sqrt{n}$-separable graph G such that $\mathbf{uMv} = 1$ if and only if there exists a cycle of length 5 incident to some vertex t in G. This 5-length cycle detection problem in turn can be solved by inspecting the diagonal entry corresponding to t of the inverse of a matrix that is defined from G. Furthermore, the diagonal entry of this matrix is inherently related to the effective resistance [41]. By appropriately dynamizing the graph G and using the time bounds for the OMv problem from the conjecture, we get the conditional lower bound for separable graphs. For general graphs, the lower bound is proved in a similar way, except that the constructed graph is different and we instead use a relation between effective resistance and triangle detection problem. That is, we first reduce the OMv problem to the t-triangle detection problem such that the OMv instance satisfies $\mathbf{uMv} = 1$ if and only if there exists a triangle incident to some vertex t in the constructed G. The latter problem can again be solved by checking the diagonal entry corresponding to t of some matrix, which in turn encodes the effective resistance of between t and a properly specified vertex s.

Other Related Work. Previous work on dynamic algorithms for planar or plane graphs include: shortest paths [29, 2, 23], $s - t$ min-cuts/max-flows [23], reachability in directed graphs [45, 22, 10], (k-edge) connected components [15, 20], the best swap and the minimum spanning forest [15]. There also exist work on dynamic algorithms for $\sqrt{n}$-separable graphs, e.g., on transitive closure and $(1 + \varepsilon)$-approximation of all-pairs shortest paths [25].

As mentioned before, subsequent to our Arxiv submission, Durfee et al. [13] obtained a dynamic all-pairs effective resistances algorithm with $\tilde{O}(m^{4/5}\varepsilon^{-4})$ expected amortized update and query time, against an oblivious adversary. This algorithm uses ideas stemmed from this paper, in particular, one of their key ideas is to dynamically maintain an approximate Schur complement. If restricted to separable graphs, the running times of their algorithm are worse than ours. It is also interesting to note that for the (simpler) offline dynamic effective resistance problems, i.e., the sequence of updates and queries are given as an input, Li et al. [32] recently gave an incremental algorithm with $O(\frac{\text{poly} \log n}{\varepsilon^2})$ amortized update and query time for general graphs.

2 Basic Tools

Our algorithm is built upon two tools: *separator trees* and *approximate Schur complement.*

Separator Trees. Let G be a $\sqrt{n}$-separable graph. For an edge-induced subgraph H of G, any vertex that is incident to vertices not in H is called a *boundary vertex.* We let $\partial(H)$ denote the set of *boundary vertices* belonging to H. A hierarchical decomposition of G is obtained by recursively partitioning the graph using separators into edge-disjoint subgraphs (called regions). This decomposition is represented by a binary (decomposition) tree $\mathcal{T}(G)$, which we refer to as a *separator tree* of G. For any subgraph H of G, we use $H \in \mathcal{T}(G)$ to denote that H is a node of $\mathcal{T}(G)$ (to avoid confusion with the vertices of G, we refer to the vertices of $\mathcal{T}(G)$ as nodes). The *height* $\eta(H)$ of a node is the number of edges in the longest path between that node and a leaf. Let $S(H)$ denote a balanced separator of the subgraph H. Further details on the definition and properties of $\mathcal{T}(G)$ can be found in the full version.

(Approximate) Schur Complement (ASC). For a given connected graph $G = (V, E)$ and a set $K \subset V$ of terminals with $1 \leq |K| \leq |V| - 1$, let $N = V \setminus K$. The partition of V into N and K naturally induces the following partition of the Laplacian $\mathbf{L}(G)$ into blocks: $\mathbf{L}(G) = \left(\begin{smallmatrix} \mathbf{L}_N & \mathbf{L}_M \\ \mathbf{L}_M^T & \mathbf{L}_K \end{smallmatrix}\right)$. We remark that since G is connected and N and K are non-empty, one can show that $\mathbf{L}_N$ is invertible. We have the following definition of Schur complement.

▶ **Definition 5** (Schur Complement). The (unique) *Schur complement* of a graph Laplacian $\mathbf{L}(G)$ with respect to a terminal set K is $\mathbf{S}(G, K) := \mathbf{L}_K - \mathbf{L}_M^T \mathbf{L}_N^{-1} \mathbf{L}_M$.

It is known that the matrix $\mathbf{S}(G, K)$ is a Laplacian for some graph G' with vertex set K.

▶ **Definition 6** (Approximate Schur Complement (ASC)). Given a graph $G = (V, E, \mathbf{w})$, $K \subset V$ and its Schur complement $\mathbf{S}(G, K)$, we say that a graph $H = (K, E_H, \mathbf{w}_H)$ is a $(1 \pm \varepsilon)$*-approximate Schur complement (abbr.* $(1 \pm \varepsilon)$-ASC*)* with respect to K if $\forall \mathbf{x} \in \mathbb{R}^{|K|}$, $(1 - \varepsilon)\mathbf{x}^T \mathbf{S}(G, K)\mathbf{x} \leq \mathbf{x}^T \mathbf{L}(H)\mathbf{x} \leq (1 + \varepsilon)\mathbf{x}^T \mathbf{S}(G, K)\mathbf{x}$.

In particular, if $\mathbf{L}(H) = \mathbf{S}(G, K)$, then we say H is a 1-ASC of G w.r.t. K.

ASC can be computed efficiently as guaranteed in the following lemma.

▶ **Lemma 7** ([14]). *Fix $\varepsilon \in (0, 1/2)$ and $\gamma \in (0, 1)$, and let $G = (V, E, \mathbf{w})$ be a graph with $K \subset V$. There is an algorithm* ApproxSchur$(G, K, \varepsilon, \gamma)$ *that computes a $(1 \pm \varepsilon)$-ASC H of G with respect to K such that the following statements hold with probability at least $1 - \gamma$: (1) The graph H has $O(|K|\varepsilon^{-2} \log(n/\gamma))$ edges. (2) The total running time for computing H is $\tilde{O}(m \log^3(n/\gamma) + n\varepsilon^{-2} \log^4(n/\gamma))$.*

3 Useful Properties of Approximate Schur Complement

Approximate Schur Complement as Vertex Effective Resistance Sparsifier. To maintain effective resistances efficiently, it will be useful to consider the following notion of *vertex sparsifier* that preserves pairwise effective resistances among a set of terminals.

▶ **Definition 8** (Vertex Resistance Sparsifier (VRS)). Given a graph $G = (V, E, \mathbf{w})$ with $K \subset V$, we say that a graph $H = (K, E_H, \mathbf{w}_H)$ is an $(1 \pm \varepsilon)$*-vertex resistance sparsifier (abbr.* $(1 \pm \varepsilon)$-VRS*)* of G with respect to K if $\forall s, t \in K$, $(1 - \varepsilon)R_G(s, t) \leq R_H(s, t) \leq (1 + \varepsilon)R_H(s, t)$.

The lemma below relates ASC and VRS (see the full version for the proof.)

▶ **Lemma 9.** *Let $G = (V, E, \mathbf{w})$ be a graph with $K \subset V$. If H is an $(1 \pm \varepsilon)$-ASC of G with respect to K, then H is an $1/(1 \pm \varepsilon)$-VRS of G with respect to K.*

Transitivity and Composability of ASCs. We will prove a *transitivity* and a *composability* property of ASCs, which will enable us to compute the ASCs of all nodes of $\mathcal{T}(G)$ in a bottom-up fashion. The corresponding proofs are deferred in the full version.

▶ **Lemma 10** (Transitivity of ASCs). *If H' is an $(1 \pm \varepsilon)$-ASC of G w.r.t. K', and H is an $(1 \pm \varepsilon)$-ASC of H' w.r.t. K, where $K' \supseteq K$, then H is an $(1 \pm \varepsilon)^2$-ASC of G w.r.t. K.*

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be edge-disjoint graphs with terminals K_1 and K_2, respectively. Furthermore, assume that $(V_1 \cap V_2) \subset K_i$, for $i = \{1, 2\}$. The *merge* of G_1 and G_2 is the graph $G = (V_1 \cup V_2, E_1 \cup E_2)$ with terminals $K_1 \cup K_2$ formed by identifying the terminals in $V_1 \cap V_2$. We denote this operation by $G := G_1 \oplus G_2$.

▶ **Lemma 11** (Composition of ASCs). *Let $G := G_1 \oplus G_2$. If H_1' is an $(1 \pm \varepsilon)$-ASC of G_1 with respect to K_1, and H_2' is an $(1 \pm \varepsilon)$-ASC of G_2 with respect to K_2, then $H' := H_1' \oplus H_2'$ is an $(1 \pm \varepsilon)$-ASC of G with respect to K.*

Algorithm 1: ApproxSchurNode($H, \partial(H), \delta'$).

```
1 Set $\gamma = 1/n^3$.
2 if $H$ is a leaf then
3     Set $H' \leftarrow$ ApproxSchur($H, \partial(H), \delta', \gamma$).
4 if $H$ is a non-leaf then
5     Let $\mathrm{c}_1(H), \mathrm{c}_2(H)$ be the children of $H$.
6     Let $\mathrm{c}_i(H)'$ be the ASC of $\mathrm{c}_i(H)$, for $i = 1, 2$.
7     Set $R \leftarrow \mathrm{c}_1(H)' \oplus_\phi \mathrm{c}_2(H)'$ and $E(R) \leftarrow E(R) \cup \mathrm{X}(H)$.
8     Set $H' \leftarrow$ ApproxSchur($R, \partial(H), \delta', \gamma$).
9 return $H'$.
```

4 Dynamic Algorithms for Effective Resistances in Separable Graphs

In this section, we first present our fully dynamic algorithm for maintaining an ASC of a $\sqrt{n}$-separable graph and then show how to extend it to dynamic effective resistances algorithm. For simplicity, we assume the separator of G can be computed in $\tilde{O}(n)$ time. We defer the discussion on the general case, some implementation details and analysis to the full version.

4.1 Dynamic Approximate Schur Complement

Let $\delta \in (0, 1)$. Let $K \subset V$ be a set of terminals with $|K| \leq O(\sqrt{n})$. We give a data-structure for maintaining a $(1 \pm \delta)$-ASC of a $\sqrt{n}$-separable graph G with respect to a set K' of $\sqrt{n}$ vertices (which contains the terminal set K) that supports Insert and Delete operations as defined before. In addition, it supports the following operation:

- AddTerminal(u): Add the vertex u to the terminal set K, as long as $|K| \leq O(\sqrt{n})$.

Data Structure. We compute and maintain a balanced separator $S(G)$ of G that contains K and satisfies that $|S(G)| \leq O(\sqrt{n})$. We let $K' = S(G)$ and we will maintain a $(1 \pm \delta)$-ASC of G w.r.t. K'. By definition of boundary vertices, $K' = \partial(G)$. Let $\delta' = \frac{\delta}{c \log n + 1}$ for some constant c. In our dynamic algorithm, we will maintain a separator tree $\mathcal{T}(G)$ (see the full version) such that for each node $H \in \mathcal{T}(G)$, we maintain its separator $S(H)$ and a set $\mathrm{X}(H)$ of edges of H, which is initially empty, and an ASC H' of H w.r.t. $\partial(H)$. Throughout the updates, the set $\mathrm{X}(H)$ will denote the subset of edges which are only contained in H while contained in neither of its children. Let $\mathcal{D}(G, \delta)$ denote such a data-structure. We recompute $\mathcal{D}(G, \delta)$ every $\Theta(\sqrt{n})$ operations using the initialization below.

Initialization. We show how to efficiently compute the ASC H' for each node H from $\mathcal{T}(G)$. We do this in a bottom-up fashion by first calling Algorithm 1 on each leaf node and then on the non-leaf nodes, where ApproxSchur is the procedure from Lemma 7. In what follows, whenever we compute an ASC, we assume that procedure ApproxSchur from Lemma 7 is invoked on the corresponding subgraph and its boundary vertices, with approximation parameter δ' and error probability $\gamma = \frac{1}{n^3}$. We will also assume that all the calls to ApproxSchur are correct.

The following lemma shows that after invoking Algorithm 1 in a bottom-up fashion, we have computed the ASC for every node in $\mathcal{T}(G)$.

Algorithm 2: UPDATEAPPROXSCHUR(STACK Q).

```
1 while $Q \neq \emptyset$ do
2     Set $H \leftarrow Q.\textsc{Pop}()$.
3     Set $H' \leftarrow \textsc{ApproxSchurNode}(H, \partial(H), \varepsilon)$.
```

▶ **Lemma 12.** *Let $H \in \mathcal{T}(G)$ be a node of height $\eta(H) \geq 0$ and $\mathrm{X}(H) = \emptyset$. Then $H' =$ APPROXSCHURNODE$(H, \partial(H), \varepsilon)$ is an $(1 \pm \delta')^{\eta(H)+1}$-ASC of H with respect to $\partial(H)$.*

Proof. We proceed by induction on $\eta(H)$. For the base case, i.e., $\eta(H) = 0$, H is a leaf node. By Lemma 7 and Algorithm 1, H' is indeed a $(1 \pm \delta')$-ASC of H with respect to $\partial(H)$.

Let H be a non-leaf node, i.e. $\eta(H) > 0$. Let $\mathrm{c}_1(H), \mathrm{c}_2(H)$ and $\mathrm{c}_1'(H), \mathrm{c}_2'(H)$ be defined as in Algorithm 1. By properties (2), (3) and (4) of $\mathcal{T}(G)$ and the fact that $X(H) = \emptyset$, we have $H = \mathrm{c}_1(H) \oplus \mathrm{c}_2(H)$. By induction hypothesis, it follows that $\mathrm{c}_i(H)'$ is an $(1 \pm \delta')^{\eta(\mathrm{c}_i(H))+1}$-ASC of $\mathrm{c}_i(H)$, for $i = 1, 2$. Using Lemma 11 and since $\eta(\mathrm{c}_i(H)) + 1 = \eta(H)$, for $i = 1, 2$, we get that $R := \mathrm{c}_1(H)' \oplus \mathrm{c}_2(H)'$ is an $(1 \pm \delta')^{\eta(H)}$-ASC of H with respect to $V(R) := \partial(\mathrm{c}_1(H)) \cup \partial(\mathrm{c}_2(H))$. Now, since $V(R) \supseteq \partial(H)$ by property (4) of $\mathcal{T}(G)$ and by Lemma 7, it follows that H' is an $(1 \pm \delta')$-ASC of R with respect to $\partial(H)$. Finally, applying Lemma 10 on R and H' we get that H' is an $(1 \pm \delta')^{\eta(H)+1}$-ASC of H. ◀

Since $\delta' = \frac{\delta}{c \log n + 1}$ and $\eta(G) = O(\log n)$, the graph G' is a $(1 \pm \delta)$-ASC of G w.r.t. $\partial(G)$.

Handling Edge Insertions. We now describe the INSERT operation. Let us consider the insertion of an edge $e = (u, v)$ of weight w. We maintain a stack Q, which is initially set to empty. We then update the root node by adding (u, v) with weight w to G, and push G onto Q. During the traversal of $\mathcal{T}(G)$, our procedure maintains two pointers that point to the current node H (initially set to G) and a node N (if any exists) that represents the node for which u and v belong to different children of N, respectively. As long as we have not found such a node N, and the current node H is not a leaf, we proceed as follows.

We examine the child of H that contains both u and v (if there is more than one, then we just pick one of them). If u and v belong to the same child, say $\mathrm{c}(H)$, then we add this edge to $\mathrm{c}(H)$ and update the current node H to $\mathrm{c}(H)$. We then push H onto Q. If, however, u and v belong to different children, then we set N to be the current node H and add the edge (u, v) to $\mathrm{X}(N)$, since u and v cannot appear together in the nodes of the lower levels. At this point, this forces u and v to become boundary vertices in N and all other nodes descending from N that contain either u or v. We handle this by making use of the ADDBOUNDARY() procedure, depicted in Algorithm 4. Finally, we recompute the ASCs of the affected nodes in a bottom-up fashion using the stack Q (as shown in Algorithm 2). This procedure is summarized in Algorithm 3. We remark that for simplicity, we let Q.PUSH(H) denote the event of pushing the pointer to H to the stack Q, for any node H.

After the pre-processing step and after each insertion/deletion of an edge, our augmented separator tree $\mathcal{T}(G)$ satisfies the following invariant.

▶ **Invariant 13.** *For every edge e in the current graph G exactly one of the following two holds: (1) there is a leaf node $H \in \mathcal{T}(G)$ such that $e \in E(H)$, (2) there is an internal node $H \in \mathcal{T}(G)$ such that $e \in X(H)$.*

The following lemma guarantees that the updated graph G' (i.e., the sparsifier of the root node G) is a good estimate to the Schur complement of G with respect to the boundary, after the execution of INSERT(u, v) in Algorithm 3, and its proof is deferred to the full version.

Algorithm 3: INSERT(u, v, w).

```
1  Let $Q$ be an initially empty stack.
2  Set $E(G) \leftarrow E(G) \cup \{(u, v)\}$, $Q$.PUSH$(G)$, $H \leftarrow G$ and $N \leftarrow$ NIL.
3  while $N =$ NIL and $H$ is a non-leaf do
4      if there exists a child of $H$ that contains both $u$ and $v$ then
5          Let c$(H)$ denote any such a child.
6          Set $E(\mathrm{c}(H)) \leftarrow E(\mathrm{c}(H)) \cup \{(u, v)\}$.
7          Set $H \leftarrow \mathrm{c}(H)$.
8          $Q$.PUSH$(H)$.
9      else
10         Set $N \leftarrow H$.
11         Set $\mathrm{X}(N) \leftarrow \mathrm{X}(N) \cup \{(u, v)\}$.
12         ADDBOUNDARY$(u, N)$, ADDBOUNDARY$(v, N)$.
13 UPDATEAPPROXSCHUR$(Q)$.          // Update the ASCs of the nodes in $Q$
```

Algorithm 4: ADDBOUNDARY(u, N).

```
1  Let $Q$ be an initially empty stack.
2  while $N =$ NIL do
3      if $u \notin \partial(H)$ then
4          Set $\partial(H) \leftarrow \partial(H) \cup \{u\}$.
5          $Q$.PUSH$(H)$.
6          if $H$ is a non-leaf then
7              Let $c(H)$ be the unique child that contains $u$.
8              Set $H \leftarrow \mathrm{c}(H)$.
9      if $H$ is a leaf then
10         Set $H \leftarrow$ NIL.
11 UPDATEAPPROXSCHUR$(Q)$.          // Update the ASCs of the nodes in $Q$
```

▶ **Lemma 14.** *Let G' be the updated sparsifier of the root node G, after the insertion of edge (u, v). Then G' is an $(1 \pm \delta)$-ASC of G with respected to $\partial(G)$.*

Handling Terminal Additions to the Boundary. We now describe the ADDTERMINAL(u) operation. It is implemented by simply invoking ADDBOUNDARY(u, G), where G is the root of $\mathcal{T}(G)$. For the procedure ADDBOUNDARY(u, H), we maintain a stack Q, which is initially set to empty. As long as the current H is a node in $\mathcal{T}(G)$, we first check whether $u \in \partial(H)$. If this is the case, then we simply do nothing as the ASC H' of H with respect to $\partial(H)$ contains u. Otherwise, we add u to $\partial(H)$, and push the node H to Q. Next, if H is not a leaf-node, let c(H) be the *unique* child that contains u. We then set c(H) to be our current node H and perform the same steps as above, until we reach some leaf-node, in which case we set H to NIL. Finally, we recompute the ASCs of the affected nodes in a bottom-up fashion using the stack Q. This procedure is summarized in Algorithm 4. The correctness of this procedure can be shown similarly to the correctness of INSERT().

Handing Edge Deletions and Running Times. The operation of deleting an edge can be handled in a symmetric way as for handling edge insertions (see the full version). For all three operations (i.e., Insert, Delete, AddTerminal), the running times are guaranteed to be $\tilde{O}(\sqrt{n}/\delta^2)$. Their analysis are deferred to the full version.

4.2 Extension to Dynamic All-Pairs Effective Resistances

Our dynamic effective resistance algorithm uses the dynamic algorithm for maintaining a $(1 \pm \delta)$-ASC as a subroutine. Formally, to maintain $(1+\varepsilon)$-approximate effective resistances, we will invoke the dynamic ASC algorithm with parameter $\delta = \varepsilon/4$, to handle edge insertions/deletions, and terminal additions.

We now describe the query operation (for the case of all-pairs effective resistances). Given s and t, we start by calling AddTerminal(s) and AddTerminal(t) from the dynamic ASC data-structure. This ensures that both s and t are boundary nodes at the root node G (if they were not previously). Thus we obtain a $(1 \pm \delta)$-ASC, denoted as G', of the root node G w.r.t. $\partial(G)$ and run on G' a nearly linear time algorithm for estimating the $s-t$ effective resistance (see the full version). Let ψ be such an estimate. For the correctness, by Lemma 9, we have that G' preserves all-pair effective resistances among vertices in $\partial(G)$ of G up to an $1/(1 \pm \delta) \approx (1 \pm 2\delta)$ factor. Since we ensured that s and t are included in $\partial(G)$, the $s-t$ effective resistance is approximated within the same factor. By a known result (see the full version), it follows that the estimate ψ approximates the effective resistance between s and t in G', up to a $(1 \pm \delta)$ factor. Combining the above guarantees, we get ψ gives an $(1 \pm 2\delta)(1 \pm \delta) \leq (1 \pm \varepsilon)$-approximation to $R_G(s,t)$, by the choice of δ. The query time will be guaranteed to be $\tilde{O}(\sqrt{n}/\varepsilon^2)$. Further details are deferred to the full version.

5 Lower Bounds for Partially Dynamic Effective Resistances

We now give a conditional lower bound for incrementally maintaining the $s-t$ effective resistance in $O(\sqrt{n})$-separable graphs and prove Theorem 3. Our proof actually holds for any algorithm that maintains a $(1 + O(\frac{1}{n^{36}}))$-approximation of $s-t$ effective resistance. The lower bounds for the decremental setting and general graphs are deferred to the full version.

The reduction. We reduce the **uMv** problem (see the definition in the full version) with parameters $n_1 = n_2 := n_0$ to the $s-t$ effective resistance problem as follows. Let $\mathbf{M}$ be the $n_0 \times n_0$ Boolean matrix of the **uMv** problem. Let $n = n_0^2 + 2n_0 + 2$. Let $\kappa = 3(n-1)^6$.

Given the matrix $\mathbf{M}$, we construct a graph $G_{\mathbf{M}} = (V_{\mathbf{M}}, E)$ as follows. (1) For each pair $1 \leq i,j \leq n_0$, we create two vertices a_{ij} and b_{ij}, and add an edge (a_{ij}, b_{ij}) if and only if $M_{ij} = 1$. (2) For each row i, we create a vertex u_i and add edge (u_i, a_{ik}) for each $1 \leq k \leq n_0$. For each column j, we create a vertex v_j and add edge (v_j, b_{kj}) for each $1 \leq k \leq n_0$. This finishes the definition of $G_{\mathbf{M}}$. Note that $V_{\mathbf{M}} = \{a_{ij}, b_{ij}, 1 \leq i,j \leq n_0\} \cup \{u_i, 1 \leq i \leq n_0\} \cup \{v_j, 1 \leq j \leq n_0\}$. For any vertex $x \in V_{\mathbf{M}}$, let $\deg_{G_{\mathbf{M}}}(x)$ denote the degree of x in $G_{\mathbf{M}}$.

Now we add two new vertices t and s to $G_{\mathbf{M}}$. For any $x \in \{a_{ij}, b_{ij}, 1 \leq i,j \leq n_0\}$, add an edge (s,x) with weight $\kappa - \deg_{G_{\mathbf{M}}}(x)$. Denote the resulting graph by G and note that G contains $|V_{\mathbf{M}} \cup \{s,t\}| = n_0^2 + 2n_0 + 2 = n$ vertices.

Assume that G is started in a dynamic effective resistance data structure. We also maintain some counters in the data structure. That is, we initialize a global counter $Y := 0$. For each vertex $x \in \{u_i, 1 \leq i \leq n_0\} \cup \{v_j, 1 \leq j \leq n_0\}$, we maintain a counter $c(x)$ which is initialized to be 0. We now explain how we use this data structure to determine **uMv**.

- Once $\mathbf{u}$ arrives, for any i such that $\mathbf{u}_i = 1$, we insert an edge (t, u_i) with weight 1, increase Y and $c(u_i)$ by 1.
- Once $\mathbf{v}$ arrives, for any j such that $\mathbf{v}_j = 1$, we insert an edge (t, v_j) with weight 1, increase Y and $c(v_j)$ by 1.
- Insert an edge (s, t) with weight $\kappa - Y$. For each vertex $x \in \{u_i, 1 \leq i \leq n_0\} \cup \{v_j, 1 \leq j \leq n_0\}$, insert an edge (s, x) with weight $\kappa - c(x) - \deg_{G_{\mathbf{M}}}(x)$.
- Perform a query EffectiveResistance(s, t) to obtain the (approximate) $s - t$ effective resistance in the final graph. Let $\lambda =$ EffectiveResistance(s, t). If $\lambda \leq \frac{1}{\kappa} + \frac{Y}{\kappa^3} + \frac{Y(n_0+1)}{\kappa^5} - \frac{1}{\kappa^6}$, then return 1; otherwise, return 0.

Analysis. Note that throughout the whole sequence of updates (which are only edge insertions) and queries, the dynamic graph G is always $O(\sqrt{n})$-separable, with a balanced separator set $S := \{u_1, \cdots, u_{n_0}\} \cup \{v_1, \cdots, v_{n_0}\} \cup \{s, t\}$ of size $O(\sqrt{n})$.

We have the following lemma that shows an important property of our reduction. The proof of the lemma is deferred to the end of this section.

▶ **Lemma 15.** *For $\kappa = 3(n-1)^6$, assume that* EffectiveResistance(s, t) *returns the exact value of the $s - t$ effective resistance in the final graph G. Then the following holds: (1) If $\mathbf{uMv} = 1$, then $\lambda \leq \frac{1}{\kappa} + \frac{Y}{\kappa^3} + \frac{Y(n_0+1)}{\kappa^5} - \frac{1}{\kappa^6}$; (2) If $\mathbf{uMv} = 0$, then $\lambda > \frac{1}{\kappa} + \frac{Y}{\kappa^3} + \frac{Y(n_0+1)}{\kappa^5} - \frac{1}{\kappa^6}$.*

Note that by the above lemma, the $\mathbf{uMv}$ problem can be solved according to our estimator λ. Thus, the lower bound for the incremental setting in Theorem 3 follows by a reduction from OMv conjecture to the $\mathbf{uMv}$ problem (see [19] and the full version of the paper) and by noting that the total number of updates is $O(n_0) = O(\sqrt{n})$ and the total number of queries is 1.

In the following we prove Lemma 15. The proof is based on a connection between the 5-length cycle detection problem and the effective resistance problem.

Proof of Lemma 15. Let G denote the final graph of our reduction. Let $H := G[V_{\mathbf{M}} \cup \{t\}]$ denote the subgraph induced by vertex set $V_{\mathbf{M}} \cup \{t\}$. We observe that in the graph H, there is a cycle of length 5 containing vertex t if and only if $\mathbf{uMv} = 1$.

On the other hand, we can use our estimator λ to distinguish if H contains a 5-length cycle incident to t or not. We let $\mathbf{A} \in \mathbb{R}^{(n-1)\times(n-1)}$ denote the adjacency matrix of the graph H. Note that all entries in A are either 1 or 0.

The first claim relates the 5-length cycle detection to the trace of a matrix related to $\mathbf{A}$. Recall that we let X_{uv} denote the entry of matrix X with row index corresponding to vertex u and column index corresponding to vertex v.

▶ **Claim 16.** *Let $\mathbf{B} = \kappa \cdot \mathbf{I} - \mathbf{A}$. If H contains a 5-length cycle incident to t, then $(\mathbf{B}^{-1})_{tt} \leq \frac{1}{\kappa} + \frac{Y}{\kappa^3} + \frac{Y(n_0+1)}{\kappa^5} - \frac{1.1}{\kappa^6}$. If H does not contain a 5-length cycle incident to t, then $(\mathbf{B}^{-1})_{tt} \geq \frac{1}{\kappa} + \frac{Y}{\kappa^3} + \frac{Y(n_0+1)}{\kappa^5} - \frac{0.9}{\kappa^6}$.*

Proof. First we note that B is invertible, as it is strictly symmetric diagonally dominant. Furthermore, it holds that $\kappa \cdot \mathbf{B}^{-1} = (I - \frac{1}{\kappa} \cdot \mathbf{A})^{-1}$ and thus by the Neumann series expansion, we have $\kappa \cdot \mathbf{B}^{-1} = (I - \frac{1}{\kappa} \cdot \mathbf{A})^{-1} = \sum_{i=0}^{\infty} (-\frac{1}{\kappa})^i \cdot \mathbf{A}^i$. This further implies that

$$(\kappa \cdot \mathbf{B}^{-1})_{tt} = \mathbf{1}_t^T (\sum_{i=0}^{\infty} (-\frac{1}{\kappa})^i \cdot \mathbf{A}^i) \mathbf{1}_t = \sum_{i=0}^{\infty} (-\frac{1}{\kappa})^i \cdot \mathbf{1}_t^T (\mathbf{A}^i) \mathbf{1}_t = \sum_{i=0}^{\infty} (-\frac{1}{\kappa})^i \cdot (\mathbf{A}^i)_{tt}.$$

Now observe that since $\kappa = 3(n-1)^6$, the first six terms of the above power series dominate. More precisely, note that $(\mathbf{A}^i)_{tt}$ is the number of i-length paths from t to t, which is at most $(n-1)^i$. Thus $\sum_{i=6}^{\infty} |(-\frac{1}{\kappa})^i \cdot (\mathbf{A}^i)_{tt}| \leq \sum_{i=6}^{\infty} \frac{1}{\kappa^i}(\mathbf{A}^i)_{tt} \leq \sum_{i=6}^{\infty} \frac{1}{\kappa^i}(n-1)^i \leq \frac{0.9}{\kappa^5}$.

Now observe that $(\mathbf{A}^0)_{tt} = \mathbf{I}_{tt} = 1$; that $\mathbf{A}_{tt} = 0$ since H is a simple graph; that $(\mathbf{A}^2)_{tt} = \deg_H(t) = Y$, where the last equation follows from the definition of Y; that $(\mathbf{A}^3)_{tt} = 0$ since there is no triangle containing t; and that $(\mathbf{A}^4)_{tt} = \sum_{w:(w,t)\in E} \sum_{x:(x,w)\in E} 1 = \sum_{w:(w,t)\in E} \deg_{G_\mathbf{M}}(w) = \det_H(t) \cdot (n_0+1) = Y(n_0+1)$. Therefore,

- If H contains a 5-length cycle incident to t, then $(\mathbf{A}^5)_{tt} \geq 2$, and thus $(\kappa \cdot \mathbf{B}^{-1})_{tt} \leq 1 + \frac{Y}{\kappa^2} + \frac{Y(n_0+1)}{\kappa^4} - \frac{2}{\kappa^5} + \frac{0.9}{\kappa^5} = 1 + \frac{Y}{\kappa^2} + \frac{Y(n_0+1)}{\kappa^4} - \frac{1.1}{\kappa^5}$
- If H has no 5-length cycle incident to t, then $(\mathbf{A}^5)_{tt} = 0$, and thus $(\kappa \cdot \mathbf{B}^{-1})_{tt} \geq 1 + \frac{Y}{\kappa^2} + \frac{Y(n_0+1)}{\kappa^4} - \frac{0.9}{\kappa^5}$

This completes the proof of the claim. ◀

The following claim relates $s-t$ effective resistance to $\mathbf{B}^{-1}$. The proof almost follows from Lemma 23 in [41] (see also the full version for the proof).

▶ **Claim 17.** *Let $\Lambda = \mathcal{E}_G(s,t)$ and $\mathbf{B} = \kappa \cdot \mathbf{I} - \mathbf{A}$. Then it holds that $\Lambda = (\mathbf{B}^{-1})_{tt}$.*

Finally, by the above two claims, if $\mathbf{uMv} = 1$, then H contains a 5-length cycle incident to t, and thus $\Lambda = (\mathbf{B}^{-1})_{tt} \leq \frac{1}{\kappa} + \frac{Y}{\kappa^3} + \frac{Y(n_0+1)}{\kappa^5} - \frac{1.1}{\kappa^6}$; if $\mathbf{uMv} = 0$, then H does not contain any 5-length cycle incident to t, and thus $\Lambda = (\mathbf{B}^{-1})_{tt} \geq \frac{1}{\kappa} + \frac{Y}{\kappa^3} + \frac{Y(n_0+1)}{\kappa^5} - \frac{0.9}{\kappa^6}$. The statement of the lemma then follows from our assumption that $\lambda = \Lambda$.

Note that our lower bound actually holds if λ is a $1 + \frac{1}{\kappa^6} = 1 + O(\frac{1}{n^{36}})$-approximation of Λ, by the above analysis and the inequality $\frac{1}{\kappa^6}(\frac{1}{\kappa} + \frac{Y}{\kappa^3} + \frac{Y(n_0+1)}{\kappa^5} - \frac{0.9}{\kappa^6}) < \frac{0.1}{\kappa^6}$. ◀

References

1 Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In *Proc. of the 57th FOCS*, pages 477–486, 2016.

2 Ittai Abraham, Shiri Chechik, and Cyril Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *Proc. of the 44th STOC*, pages 1199–1218, 2012.

3 Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In *Proc. of the 57th FOCS*, pages 335–344, 2016.

4 Nima Anari and Shayan Oveis Gharan. Effective-resistance-reducing flows, spectrally thin trees, and asymmetric tsp. In *Proc. of the 56th FOCS*, pages 20–39, 2015.

5 Daniel K Blandford, Guy E Blelloch, and Ian A Kash. Compact representations of separable graphs. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 679–688. Society for Industrial and Applied Mathematics, 2003.

6 James R Bunch and John E Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.

7 Dehua Cheng, Yu Cheng, Yan Liu, Richard Peng, and Shang-Hua Teng. Efficient sampling for gaussian graphical models via spectral sparsification. In *Conference on Learning Theory*, pages 364–390, 2015.

8 Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proc. of the 43rd STOC*, pages 273–282, 2011.

9 Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Shen Chen Xu. Solving SDD linear systems in nearly $m\log^{1/2} n$ time. In *Proc. of the 46th STOC*, pages 343–352, 2014.

10 Krzysztof Diks and Piotr Sankowski. Dynamic plane transitive closure. In *European Symposium on Algorithms*, pages 594–604. Springer, 2007.

11 Michael Dinitz, Robert Krauthgamer, and Tal Wagner. Towards resistance sparsifiers. In *Proc. of the 18th APPROX*, pages 738–755, 2015.

12 Peter G Doyle and J Laurie Snell. *Random Walks and Electric Networks.* Carus Mathematical Monographs. Mathematical Association of America, 1984.

13 David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng. Fully Dynamic Effective Resistances. *ArXiv e-prints*, apr 2018. `arXiv:1804.04038`.

14 David Durfee, Rasmus Kyng, John Peebles, Anup B Rao, and Sushant Sachdeva. Sampling random spanning trees faster than matrix multiplication. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 730–742. ACM, 2017.

15 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification. i. planary testing and minimum spanning trees. *J. Comput. Syst. Sci.*, 52(1):3–27, 1996.

16 Gramoz Goranci, Monika Henzinger, and Pan Peng. The power of vertex sparsifiers in dynamic graph algorithms. In *Proc. of the 25th ESA*, volume 87, pages 45:1–45:14, 2017.

17 Gramoz Goranci, Monika Henzinger, and Pan Peng. Dynamic effective resistances and approximate schur complement on separable graphs. *CoRR*, abs/1802.09111, 2018. `arXiv:1802.09111`.

18 Prahladh Harsha, Thomas P Hayes, Hariharan Narayanan, Harald Räcke, and Jaikumar Radhakrishnan. Minimizing average latency in oblivious routing. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 200–207. Society for Industrial and Applied Mathematics, 2008.

19 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proc. of the 47th STOC*, pages 21–30, 2015.

20 Jacob Holm, Giuseppe F Italiano, Adam Karczmarz, Jakub Łacki, Eva Rotenberg, and Piotr Sankowski. Contracting a planar graph efficiently. In *25th European Symposium on Algorithms, ESA 2017*. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2017.

21 Oscar H Ibarra, Shlomo Moran, and Roger Hui. A generalization of the fast lup matrix decomposition algorithm and applications. *Journal of Algorithms*, 3(1):45–56, 1982.

22 Giuseppe F Italiano, Adam Karczmarz, Jakub Łącki, and Piotr Sankowski. Decremental single-source reachability in planar digraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1108–1121. ACM, 2017.

23 Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proc. of the 43rd STOC*, pages 313–322, 2011.

24 Arun Jambulapati and Aaron Sidford. Efficient $\tilde{O}(n/\varepsilon)$ spectral sketches for the laplacian and its pseudoinverse. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2487–2503. SIAM, 2018.

25 Adam Karczmarz. Decrementai transitive closure and shortest paths for planar digraphs and beyond. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 73–92. SIAM, 2018.

26 Ken-ichi Kawarabayashi and Bruce Reed. A separator theorem in minor-closed classes. In *Proc. of the 51st FOCS*, pages 153–162. IEEE, 2010.

27 Jonathan A Kelner, Gary L Miller, and Richard Peng. Faster approximate multicommodity flow using quadratically coupled flows. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1–18. ACM, 2012.

28 Douglas J Klein and Milan Randić. Resistance distance. *Journal of mathematical chemistry*, 12(1):81–95, 1993.

29 Philip N. Klein and Sairam Subramanian. A fully dynamic approximation scheme for shortest paths in planar graphs. *Algorithmica*, 22(3):235–249, 1998.

30 Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. Sparsified cholesky and multigrid solvers for connection laplacians. In *Proc. of the 48th STOC*, 2016.

31 Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians - fast, sparse, and simple. In *Proc. of the 57th FOCS*, pages 573–582, 2016.

32 Huan Li, Stacy Patterson, Yuhao Yi, and Zhongzhi Zhang. Maximizing the Number of Spanning Trees in a Connected Graph. *ArXiv e-prints*, 2018. `arXiv:1804.02785`.

33 Huan Li and Zhongzhi Zhang. Kirchhoff index as a measure of edge centrality in weighted networks: Nearly linear time algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2377–2396. SIAM, 2018.

34 Richard J. Lipton, Donald J. Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16(2):346–358, 1979.

35 Richard J Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

36 Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *Proc. of the 54th FOCS*, pages 253–262, 2013.

37 Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *Proc. of the 57th FOCS*, pages 593–602, 2016.

38 Aleksander Mądry, Damian Straszak, and Jakub Tarnawski. Fast generation of random spanning trees and the effective resistance metric. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 2019–2036. Society for Industrial and Applied Mathematics, 2015.

39 Gary L. Miller and Richard Peng. Approximate maximum flow on separable undirected graphs. In *Proc. of the 24th SODA*, pages 1151–1170, 2013.

40 Gary L Miller, Shang-Hua Teng, William Thurston, and Stephen A Vavasis. Separators for sphere-packings and nearest neighbor graphs. *Journal of the ACM (JACM)*, 44(1):1–29, 1997.

41 Cameron Musco, Praneeth Netrapalli, Aaron Sidford, Shashanka Ubaru, and David P. Woodruff. Spectrum Approximation Beyond Fast Matrix Multiplication: Algorithms and Hardness. *LIPIcs*, 94:8:1–8:21, 2018.

42 Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 509–517. IEEE Computer Society, 2004.

43 Aaron Schild. An almost-linear time algorithm for uniform random spanning tree generation. *arXiv preprint arXiv:1711.06455*, 2017.

44 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.*, 40(6):1913–1926, 2011.

45 Sairam Subramanian. A fully dynamic data structure for reachability in planar digraphs. In *Proc. of the 1st ESA*, pages 372–383, 1993.

46 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 887–898. ACM, 2012.

Buffered Count-Min Sketch on SSD: Theory and Experiments

Mayank Goswami
Queens College, City University of New York
mayank.goswami@qc.cuny.edu

Dzejla Medjedovic
International University of Sarajevo
dzmedjedovic@ius.edu.ba

Emina Mekic
Sarajevo School of Science and Technology
emina.mekic@stu.ssst.edu.ba

Prashant Pandey
Stony Brook University, New York
ppandey@cs.stonybrook.edu

Abstract

Frequency estimation data structures such as the count-min sketch (CMS) have found numerous applications in databases, networking, computational biology and other domains. Many applications that use the count-min sketch process massive and rapidly evolving data sets. For data-intensive applications that aim to keep the overestimate error low, the count-min sketch becomes too large to store in available RAM and may have to migrate to external storage (e.g., SSD.) Due to the random-read/write nature of hash operations of the count-min sketch, simply placing it on SSD stifles the performance of time-critical applications, requiring about 4-6 random reads/writes to SSD per estimate (lookup) and update (insert) operation.

In this paper, we expand on the preliminary idea of the buffered count-min sketch (BCMS) [Eydi et al., 2017], an SSD variant of the count-min sketch, that uses hash localization to scale efficiently out of RAM while keeping the total error bounded. We describe the design and implementation of the buffered count-min sketch, and empirically show that our implementation achieves 3.7×-4.7× speedup on update and 4.3× speedup on estimate operations compared to the traditional count-min sketch on SSD.

Our design also offers an asymptotic improvement in the external-memory model over the original data structure: r random I/Os are reduced to 1 I/O for the estimate operation. For a data structure that uses k blocks on SSD, w as the word/counter size, r as the number of rows, M as the number of bits in the main memory, our data structure uses kwr/M amortized I/Os for updates, or, if $kwr/M > 1$, 1 I/O in the worst case. In typical scenarios, kwr/M is much smaller than 1. This is in contrast to $O(r)$ I/Os incurred for each update in the original data structure.

Lastly, we mathematically show that for the buffered count-min sketch, the error rate does not substantially degrade over the traditional count-min sketch. Specifically, we prove that for any query q, our data structure provides the guarantee: $\Pr[\text{Error}(q) \geq n\epsilon(1 + o(1))] \leq \delta + o(1)$, which, up to $o(1)$ terms, is the same guarantee as that of a traditional count-min sketch.

2012 ACM Subject Classification Theory of computation → Data structures and algorithms for data management, Theory of computation → Streaming models, Theory of computation → Database query processing and optimization (theory)

Keywords and phrases Streaming model, Count-min sketch, Counting, Frequency, External memory, I/O efficiency, Bloom filter, Counting filter, Quotient filter

© Mayank Goswami, Dzejla Medjedovic, Emina Mekic, and Prashant Pandey;
licensed under Creative Commons License CC-BY

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.41

Acknowledgements We gratefully acknowledge support from CNS 1408695, CCF 1439084, IIS 1247726, CCF 1725543, SPX, CCF 1716252, CCF 1617618, CNS-1755615, CCF 1755791, and from Sandia National Laboratories.

1 Introduction

Applications that generate and process *massive data streams* are becoming pervasive [3, 18, 19, 14, 25] across many domains in computer science. Common examples of streaming data sets include financial markets, telecommunications, IP traffic, sensor networks, textual data, etc [3, 10, 26, 7]. Processing fast-evolving and massive data sets poses a challenge to traditional database systems, where commonly the application stores all data and subsequently does queries on it. In the streaming model [3], the data set is too large to be completely stored in the available memory, so every item is seen and processed once — an algorithm in this model performs only one scan of data, and uses sublinear local space.

The streaming scenario exhibits some limitations on the types of problems we can solve with such strict time and space constraints. A classic example is the heavy hitter problem `HH(k)` on the stream of pairs (a_t, c_t), where a_t is the item identifier, and c_t is the count of the item at timeslot t, with the goal of reporting all items whose frequency is at least n/k, $n = \sum_{t=1}^{T} c_t$. The general version of the problem with the exception of when k is a small constant[1], can not be exactly solved in the streaming model [22, 26], but the approximate version of the problem, ϵ-`HH(k)`, where all items of the frequency at least $n/k - \epsilon n$ are reported, and an item with larger error might be reported with small probability δ, is efficiently solved with the count-min sketch [11] data structure. The count-min sketch accomplishes this in $O(\ln(1/\delta)/\epsilon)$ space, usually far below linear space in most applications.

The count-min sketch [11] has been extensively used to answer heavy hitters, top k queries and other *popularity measure queries*, the central problems in the streaming context, where we are interested in extracting the essence from an impractically large amount of data. Common applications include displaying the list of bestselling items, the most clicked-on websites, the hottest queries on the search engine, most frequently occurring words in a large text, and so on [24, 19, 27].

The count-min sketch (CMS) is a hashing-based, probabilistic, and lossy representation of a multiset, that is used to answer the count of an item a (number of times a appears in a stream). It has two error parameters: 1) ϵ, which controls the overestimation error, and 2) δ, which controls the failure probability of the algorithm. The CMS provides the guarantee that the estimation error for any item a is more than ϵn with probability at most δ. If we set $r = \ln(1/\delta)$ and $c = e/\epsilon$, the CMS is implemented using r hash functions as a 2D array of dimensions $r \cdot c$.

When ϵ and δ are constants, the total overestimate grows proportionately with n, the size of the count-min sketch remains small, and the data structure easily fits in smaller and faster levels of memory. For some applications, however, the allowed estimation error of ϵn is too high when ϵ is fixed. Consider an example of $n = 2^{30}$, where $\delta = 0.01$ and $\epsilon = 2^{-26}$, hence the overestimate is 16, and the total data structure size of 3.36GB, provided each counter uses 4 bytes. However, if we double the data set size, then the total overestimate also doubles to 32 if ϵ stays the same. On the other hand, if we want to maintain the fixed overestimate of 16, then the data structure size doubles to 6.72GB.

[1] When $k \approx 2$ this problem goes by the name of majority element.

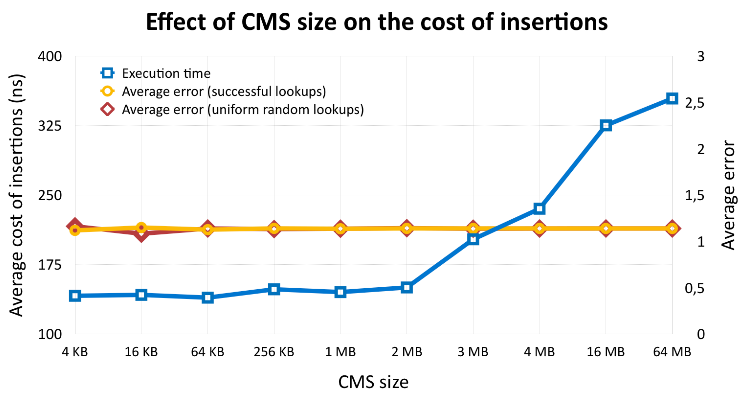


Figure 1 The effect of increasing the count-min sketch size on the update operation cost in RAM.

In this paper, we expand on the preliminary idea of the buffered count-min sketch (BCMS) [13], an SSD variant of the traditional count-min sketch data structure, that scales efficiently to large data sets while keeping the total error bounded. Our work expands on the previous work by introducing a detailed design, implementation, and experiments, as well as mathematical analysis of the new data structure (our original paper [13], which, to the best of our knowledge is the only attempt thus far to scale the count-min sketch to SSD, contains only the outline of the data structure). Our analysis is performed in the external-memory model [1], which emphasizes the cost of I/O operations over CPU computation. In the external-memory model, the unit cost is a block transfer of size B between the disk of infinite size and the main memory of size M (for most input sizes N, $M << N$.)

To demonstrate the issues arising from a growing count-min sketch and storing it in lower levels of memory, we run a mini in-RAM experiment for count-min sketch sizes 4KB-64MB. In Figure 1, we see that to maintain the same error, the cost of update will increase as the data structure is being stored in the lower levels of memory, even though we keep the number of hash functions fixed for all data structure sizes. The appropriate peak in the cost is visible at the border of L2 and L3 cache (at 3MB).

Asymptotically, storing the unmodified count-min sketch on SSD or a disk is inefficient, given that each estimate and update operation needs r hashes, which results in $O(r)$ random reads/writes to SSD, far below the desired throughput for most time-critical streaming applications.

Another context where we see the CMS becoming large even when ϵ is fixed is in some text applications, where the number of elements inserted in the sketch is quadratic in the original text size. For instance, [17] uses the CMS to record distributional similarity on the web, where each pair of words is inserted as a single item into the CMS, and 90GB of text requires a CMS of 8GB.

We focus on scenarios where the allowed estimation error is sublinear in n. For example, what if we want the estimation error to be no larger than $n/\log n$, or $\sqrt{n}$? These scenarios correspond to $\epsilon = 1/\log n$ or $1/\sqrt{n}$, and now for even moderately large values of n, the count-min sketch becomes too large to fit in main memory. Given more modest condition, such as $\epsilon = o(1/M)$, where the memory is of size M, the count-min sketch is unlikely to fit in memory. We will assume that $1/n \leq \epsilon << 1/M$. Higher values of ϵ do not require the count-min sketch to be placed on disk, and lower values of ϵ mean exact counts are desired.

1.1 Results

1. We describe the design and implementation of the buffered count-min sketch, and empirically show that our implementation achieves $3.7\times - 4.7\times$ the speedup on update and $4.3\times$ speedup on estimate operations.
2. Our design also offers an asymptotic improvement in the external-memory model [1] over the original data structure: $O(r)$ random I/Os are reduced to 1 I/O for estimate. For a data structure that uses k blocks on SSD, w as the word/counter size, r as the number of rows, M as the number of bits in main memory, our data structure uses kwr/M amortized I/Os for updates, or, if $kwr/M > 1$, 1 I/O in the worst case. In typical scenarios, $kwr/M << 1$. This is in contrast to $O(r)$ I/Os incurred for each update in the original data structure.
3. We mathematically show that for the buffered count-min sketch, the error rate does not substantially degrade over the original count-min sketch. Specifically, we prove that for any query q, our data structure provides the following guarantee:

$$\Pr[\text{Error}(q) \geq n\epsilon(1 + o(1))] \leq \delta + o(1).$$

2 Background

The streaming model represents many real-life situations where the data is produced rapidly and on a constant basis. For example, sensor networks [19], monitoring web traffic [23], analyzing text [17], and monitoring satellites orbiting the Earth [16], etc.

Heavy hitters, top-k queries, iceberg queries, and quantiles [25, 19, 3] are some of the most central problems in the streaming context, where we wish to extract general trends from a massive data set. The count-min sketch has proved useful in such contexts for its space-efficiency and providing count estimates [11, 18].

The count-min sketch can be well illustrated using its connection to the Bloom filter [5, 6]. Both data structures are lossy and space-efficient representations and used to reduce disk accesses in time-critical applications. The Bloom filter answers membership queries and occasionally returns false positives while the count-min sketch answers frequency queries and occasionally returns overestimates. Both data structures are hashing-based and suffer from similar issues when placed directly on SSDs or rotating disks.

There have been earlier attempts to scale Bloom filters to SSD using buffering and hash localization [8, 12]. Our paper employs similar methods to those in [8, 12]. The improvements, both in our case and in the case of the Buffered Bloom filter [8] are achieved at the expense of having an extra hash function that helps determine the page the item belongs to.

Work has also been done in designing counting filters [4, 20], such as the counting quotient filter (CQF) and its SSD variant, the cascade filter (write-optimized quotient filter) [4]. However, there is an important distinction between counting filter data structures and the count-min sketch. The CQF gives exact counts for most of the elements given that the CQF has small false-positive error. However, since errors are independent, the CQF does not offer guarantees on the overestimate. For example, two highly occurring elements in a multiset can collide with each other and both will have large overcounts. On the other hand, the count-min sketch does not give exact counts of elements but offers a guarantee that overestimate will be smaller than ϵn with a probability of δ

A similar data structure to count-min sketch is count-sketch [9]. Count sketch offers tighter error bounds than traditional count-min sketch, expressed through L^2-norm as oppose to L^1-norm. However, the error is two-sided, and gains in accuracy require the factor of ϵ

blowup in space. The count-sketch can be advisable where the smaller ϵ is desired, however, that would require a much larger data structure. Also, the count-min sketch is more widely used and applicable and this is why we choose to analyze its SSD performance. One can also hypothesize that extensions of the count-sketch to disk would benefit from the same hash localization and buffering techniques as did the count-min sketch given their almost identical structure.

2.1 External-Memory Model

We use the external-memory model or disk-acces machine (DAM) model [1] to analyze the on-SSD performance of our data structure. DAM model captures the essential feature of modern computers, where the CPU computation is orders-of-magnitude cheaper than moving data between different levels of memory. This deems the cost of I/O transfers the main bottleneck in many data-intensive applications. In the DAM model, memory is made up of two levels, main memory of size M and disk of infinite size, and data is transferred between the two levels using blocks of size B, where usually $M = \Theta(B^2)$. Once data is in the memory, all computations are free, and the performance is measured solely by the number of disk transfers performed. Even though the DAM model only shows the communication between RAM and disk, it is a useful analogy for any two levels of memory where one is small and fast and the other one is large and slow (i.e., different cache levels). Therefore, the problem size need not be that large for the I/O effects to kick in and the DAM model to be applicable.

2.2 Count-Min Sketch: Preliminaries

In the streaming model, we are given a stream A of pairs (a_i, c_i), where a_i denotes the item identifier (e.g., IP address, stock ID, product ID), and c_i denotes the count of the item. Each pair $X_i = (a_i, c_i)$ is an item within a stream of length T, and the goal is to record total sum of frequencies for each particular item a_i.

For a given estimation error rate ϵ and failure probability δ, define $r = \ln(1/\delta)$ and $c = e/\epsilon$. The count-min sketch is represented via a 2D matrix with c buckets (columns), r rows, implemented using r hash functions (one hash function per row). CMS has two operations: `UPDATE(`a_i`)` and `ESTIMATE(`a_i`)`, the respective equivalents of `insert` and `lookup`, and they are performed as follows:

1. `UPDATE(`a_i`)` inserts the pair by computing r hash functions on a_i and incrementing appropriate slots determined by the hashes by the quantity c_i. That is, for each hash function h_j, $1 \leq j \leq r$, we set $CMS[j][h_j(a_i)] = CMS[j][h_j(a_i)] + c_i$. Note that in this paper, we use $c_i = 1$, so every time an item is updated, it is just incremented by 1.
2. `ESTIMATE(`a_i`)` reports the frequency of a_i which can be an overestimate of the true frequency. It does so by calculating r hashes and taking the minimum of the values found in appropriate cells. In other words, we return $min_{1 \leq j \leq r}(CMS[j][h_j(a_i)])$. Because different elements can hash to the same cells, the count-min sketch can return the overestimated (never underestimated) value of the count, but in order for this to happen, a collision needs to occur in each row. The estimation error is bounded; the data structure guarantees that for any particular item, the error is within the range ϵn, with probability at least $1 - \delta$, i.e., $Pr[\text{Error}(q) \geq \epsilon n] \leq \delta$.

3 Buffered Count-Min Sketch

In this section, we describe the buffered count-min sketch, an adaptation of the count-min sketch to SSD. The traditional CMS, when placed on external storage, exhibits performance issues due to random-write nature of hashing. Each update operation in the CMS requires $r = \ln(1/\delta)$ writes to different rows and columns of the CMS. On a large data structure, these writes become destined to different pages on disk, causing the update to perform $O(\ln(1/\delta))$ random SSD page writes. For high-precision CMSs, where $\delta = 0.001\% - 0.01\%$, this can be between 5-7 writes to SSD, which is unacceptable in a high-throughput scenario.

To solve this problem, we implement, analyze, and empirically test the data structure presented in [13] that outlines three adaptations to the original data structure:

1. Partitioning the CMS into pages and column-first layout: We logically divide the CMS on SSD into pages of block size B. CMS with r rows, c columns, cell size w, and a total of $S = cr$ w-bit counters, contains k pages $P_1, P_2, P_3, \ldots, P_k$, where $k = S/B$ and each page spans contiguous B/r columns [2]: P_i spans columns $[B(i-1)/r + 1, Bi/r]$. To improve cache-efficiency, the CMS is laid out on disk in column-first order which allows each logical page to be laid out sequentially in memory. Thus, each read/write of a logical page requires at most 2 I/Os.
2. Hash localization: We direct all hashes of an element to a single logical page in the CMS. The page is determined by an additional hash function $h_0 : [1, k]$. The subsequent r hash functions map to the columns inside the corresponding logical page, i.e., the range of $h_1, h_2, \ldots, h_r$ for an element e is $[B(h_0(e) - 1)/r + 1, Bh_0(e)/r]$. This way, we direct all updates and reads related to an element to one logical page.
3. Buffering: When an update operation occurs, the hashes produced for an element are first stored inside an in-memory buffer. The buffer is partitioned into sub-buffers of equal size $S_1, S_2, \ldots, S_k$, and they directly correspond to logical pages on disk in that S_i stores the hashes for updates destined for page P_i. Each element first hashes using h_0, which determines in which sub-buffer the hashes will be temporarily stored for this element. Once the sub-buffer S_i becomes full, we read the page P_i from the CMS, apply all updates destined for that page, and write it back to disk. The capacity of a sub-buffer is M/k hashes, which is equivalent to M/kwr elements so the cost of an update becomes $kwr/M << 1$ I/O.

Algorithm 1 shows the pseudocode for `UPDATE(`a_i`)` operation and Algorithm 2 shows the pseudocode for `ESTIMATE(`a_i`)` operation. We use `murmurhash` [2] as our hash function. In the buffered count-min sketch, there is no buffering in `ESTIMATE(`a_i`)` operation and it is optimized for the worst-case single lookups and mixed (i.e., simultaneous updates and estimates) workloads. The `ESTIMATE(`a_i`)` first computes the correct sub-buffer using h_0, and flushes the corresponding sub-buffer to SSD page in case some updates were present. Once it applies the necessary changes to the page, it reads the corresponding CMS cells specified by r hashes and returns the minimum estimate.

4 Analysis of Buffered Count-Min Sketch

In this section, we show that the buffering and hash localization do not substantially degrade the error guarantee of the buffered count-min data structure. Fix a failure probability $0 < \delta < 1$ and let $0 < \epsilon(n) < 1$ be the function of n controlling the estimation error. Let

[2] For most practical configurations the page size B is larger than the number of rows r.

Algorithm 1 Buffered Count-Min Sketch - UPDATE function.

```
Require: key, r
subbufferIndex$_i$ :=murmur$_0$(key);
for i:=1 to r do
   hashes[i] :=murmur$_i$(key);
end for
AppendToBuffer(hashes,subbufferIndex);

if isSubbufferFull(subbufferIndex) then
   bcmsBlock :=readDiskPage(subbufferIndex);
   for each entry in Subbuffer[subbufferIndex] do
       for each index in entry do
           pageStart :=calculatePageStart(subbufferIndex);
           offset :=pageStart + entry[index];
           bcmsBlock[offset][index]++;
       end for
   end for
   writeBcmsPageBackToDisk(bcmsBlock);
   clearBuffer(subbufferIndex);
end if
```

Algorithm 2 Buffered Count-Min Sketch - ESTIMATE function.

```
Require: key, k
subbufferIndex$_i$ :=murmur$_0$(key);
pageStart :=calculatePageStart(subbufferIndex);
bcmsBlock :=readDiskPage(subbufferIndex);

 if isSubbufferNotEmpty(subbufferIndex) then
   for each entry in Subbuffer[subbufferIndex] do
       for each index in entry do
           offset :=pageStart + entry[index];
           bcmsBlock[offset][index]++;
       end for
   end for
   clearBuffer(subbufferIndex);
end if

for i:=1 to k do
   value :=murmur$_i$(key);
   offset :=pageStart + value;
   estimation :=bcmsBlock[offset][i - 1];
   estimates[i] :=estimation;
end for
writeBcmsPageBackToDisk(bcmsBlock);
return min(estimates)
```

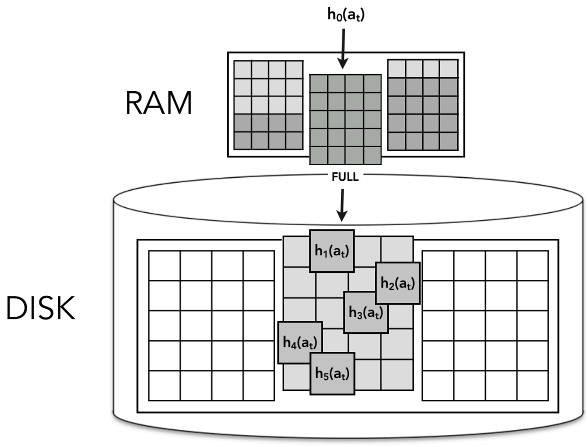


Figure 2 `UPDATE` operation in the buffered count-min sketch. In-RAM buffer is divided into sub-buffers and when a sub-buffer is full all updates are flushed to the corresponding page on disk.

$r = \ln(1/\delta)$ and $c = e/\epsilon$. The traditional count-min sketch uses $S = cr = (e/\epsilon)\ln(1/\delta)$ counters/words of space. Recall that for our purposes, $1/n \le \epsilon(n) << 1/M$.

Let $k = S/B$ be the number of blocks occupied by the buffered count-min sketch. We assume a block can hold B counters. Our analysis will assume the following mild conditions:

Assumption 1: We assume that n is sufficiently larger than the number of blocks k, $n = \omega(k(\log k)^3)$ suffices. Since k depends inversely on $\epsilon(n)$, this assumption essentially means that $\epsilon(n) = \omega(1/n)$.

Assumption 2: We assume that $\lim_{n\to\infty} \epsilon(n) = 0$.

Both conditions are satisfied, e.g., when $\epsilon(n) = 1/\log n$ or $1/n^c$ for any $c < 1$.

For brevity, we will drop the dependence of $\epsilon(n)$ on n, and write the error rate as just ϵ, however it is important to note that ϵ is not a constant.

▶ **Theorem 1.** *The Buffered-Count-Min-Sketch is a data structure that uses k blocks of space on disk and for any query q,*

1. *returns `ESTIMATE(q)` in 1 I/O and performs `UPDATE(q)` in kwr/M I/Os amortized, or, if $kwr/M > 1$, in one I/O worst case.*
2. *Let `Error(q)` = `ESTIMATE(q)` - `TrueFrequency(q)`. Then for any $C \ge 1$,*

$$Pr[\texttt{Error}(q) \ge n\epsilon(1 + \sqrt{(2(C+1)k\log k)/n})] \le \delta + O((\epsilon B/e)^C).$$

Remark: By Assumption 1, $\sqrt{(2(C+1)k\log k)/n}$ is $o(1)$ (in fact, it is $o(1/\log k)$). By Assumption 2, $(\epsilon B/e)^C$ is $o(1)$. Thus we claim that the buffered count-min-sketch gives almost the same guarantees as a traditional count-min sketch, while obtaining a factor r speedup in queries. The guarantee for estimates taking 1 I/O is apparent from construction, as only one block needs to be loaded[3].

The proof is a combination of the classical analysis of CMS and the maximum load of balls in bins when the number of bins is much smaller than the number of balls. Also, note that unlike the traditional CMS, the errors for a query q in different rows are no longer independent (in fact, they are positively correlated: a high error in one row implies more elements were hashed by h_0 to the same bucket as q).

[3] In practice, we may need 2 I/Os due to block-page alignment, but never more than 2.

The hash function h_0 maps into k buckets, each having size B (and so we will also call them blocks). Each bucket can be thought of as a $r \times B/r$ matrix. Note that $r = \ln(1/\delta)$, and $B/r = e/(\epsilon k)$. We assume that h_0 is a perfectly random hash function, and, abusing notation, identify a bucket/block with a bin, where h_0 assigns elements (balls) to one of the k buckets (bins).

In this scenario we use Lemma 2(b) from [21] and adapt it to our setting.

▶ **Lemma 2.** *(Lemma 2(b) from [21]) Let $B(n,p)$ denote a Binomial distribution with parameters n and p, and $q = 1-p$. If $t = np + o((pqn)^{2/3})$ and $x := \frac{t-np}{\sqrt{pqn}}$ tends to infinity, then*

$$Pr[B(n,p) \geq t] = e^{-x^2/2 - \log x - \frac{1}{2}\log \pi + o(1)}.$$

Let $M(n,k)$ denote the maximum number of elements that fall into a bucket, when hashed by h_0.

▶ **Lemma 3.** *Let $C \geq 1$ and $t = n/k + \sqrt{2(C+1)\frac{n \log k}{k}}$. Then*

$$Pr[M(n,k) \leq t] \geq 1 - 1/k^C.$$

Proof. We first check that t satisfies the conditions of Lemma 2. Since h_0 is uniform, $p = 1/k$ (i.e., each bucket is equally probable), and $np = n/k$. We need to check that the extra term in t, $\sqrt{2(C+1)\frac{n \log k}{k}}$ is $o((n(1-1/k)/k)^{2/3})$. This is precisely the condition that $n = \omega(k(\log k)^3)$ (Assumption 1).

Next we apply Lemma 2. In our case,

$$x = \sqrt{\frac{2(C+1)n \log k/k}{n(1-1/k)/k}} = \sqrt{2(C+1)\log k(1+1/k-1)},$$

Now by assumption 2, $\epsilon(n)$ goes to zero as n goes to infinity, and so $k \propto 1/\epsilon(n)$ goes to infinity, and therefore x goes to infinity as n goes to infinity. Thus we have that the number of elements in any particular bucket (which follows a $B(n,1/k)$ distribution) is larger than t with probability $e^{-x^2/2-\log x - \frac{1}{2}\log \pi + o(1)} \leq e^{-x^2/2}$. Putting in $x = \sqrt{2(C+1)\log k(1+\frac{1}{k-1})}$, we get $x^2/2 = (C+1)\log k(1+1/(k-1) \geq (C+1)\log k$, and thus the probability is at most $e^{-(C+1)\log k} = 1/k^{C+1}$.

Thus the probability that the maximum number of balls in a bin is more than t is bounded (by the union bound) by $k * (1/k)^{C+1} = (1/k)^C$, and the lemma is proved. ◀

Now that we know that with probability as least $1 - 1/k^C$, no bucket has more than t elements, we observe that a bucket serves as a "mini" CMS for the elements that hash to it. In other words, let $n(q)$ be the number of elements that hash to the same bucket as q under h_0. The expected error in the ith row of the mini-CMS for q (the entry for which is contained inside the bucket of q), is $\mathbb{E}[\texttt{Error}_i(q)] = n(q)/(B/r) = n(q)\epsilon k/e$.

By Markov's inequality $\Pr[\texttt{Error}_i(q) \geq n(q)k\epsilon] \leq 1/e$.

Let $\alpha = t\epsilon k/e = (n/k + \sqrt{(2(C+1)n \log k)/k})\epsilon k/e = (n\epsilon/e)(1 + \sqrt{(2(C+1)k \log k)/n})$. We now compute the bound on the final error (after taking the min) as follows.

$$\begin{aligned} Pr(\text{Error}(q) \geq e\alpha) &= Pr(\text{Error}_i(q) \geq e\alpha \quad \forall i \in \{1, \cdots, r\}) \\ &= Pr(\text{Error}_i(q) \geq e\alpha \quad \forall i| \quad n(q) \leq t) Pr(n(q) \leq t) \\ &+ Pr(\text{Error}_i(q) \geq e\alpha \quad \forall i| \quad n(q) \geq t) Pr(n(q) \geq t) \\ &\leq (1/e)^r + (1/k)^C \\ &= \delta + 1/k^C, \end{aligned}$$

where the second last equality follows from Markov's inequality on $\text{Error}_i(q)$ and Lemma 3. Finally, by observing that for a fixed δ, $k = O(e/B\epsilon)$, the proof of the theorem is complete.

5 Evaluation

In this section, we evaluate our implementation of the buffered count-min sketch. We compare the buffered count-min sketch against the (traditional) count-min sketch. We evaluate each data structure on two fundamental operations, update and estimate. We evaluate estimate operation for a set of elements chosen uniformly at random.

In our evaluation, we address the following questions about how the performance of the buffered count-min sketch compares to the count-min sketch:

1. How does the update throughput in the buffered count-min sketch compare to the count-min sketch on SSD?
2. How does the estimate throughput in the buffered count-min sketch compare to the count-min sketch on SSD?
3. What is the effect of hash localization in the buffered count-min sketch on the frequency overestimate compared to the frequency overestimate in the count-min sketch?
4. What is the effect of changing the RAM-size-to-sketch-size ratio on the update and estimate performance?

5.1 Experimental setup

To answer the above questions, we evaluate the performance of the buffered count-min sketch and the (traditional) count-min sketch on SSD by scaling the sketch out of RAM. For SSD benchmarks, we use four different RAM-size-to-sketch-size ratios: 2, 4, 8, and 16. The RAM-size-to-sketch-size ratio is the ratio of the size of the available RAM and the size of the sketch on SSD. To do this, we fix the size of the available RAM to $\approx$ 64MB and increase the sketch size to manipulate the ratio. Note that even though 64MB is a rather modest RAM size, we are primarily interested in observing the changes in performance when the ratio between RAM and sketch on SSD changes — it is this ratio that determines the frequency of flushing, and results on a 64MB RAM size should extend to any other RAM/SSD sizes, if the ratio is preserved. The page size in all our benchmarks was set to 4096B. In all the benchmarks, we measure the throughput (operations per second) to evaluate the update and estimate performance.

To measure the update throughput, we first calculate the number of elements we can insert in the sketch using calculations described in Section 5.2. During an update operation, we generate 64-bit integers online from a uniform-random distribution using the pseudo-random number generator in C++. This way, we do not use any extra memory to store the set of integers to be added to the sketch. We then measure the total time taken to update the given set of elements in the sketch. Note that for the buffered count-min sketch, we make sure to flush all the remaining updates from the buffer to the sketch on SSD after the last update and include the time to do that in the total time.

To measure the estimate throughput, we query for the estimate of elements drawn from a uniform-random distribution and measure the throughput. The workload in the estimate benchmark simulates a real-world query workload where some elements may not be present in the sketch and the estimate operation will terminate early thereby requiring fewer I/Os.

For all the estimate benchmarks, we first perform the update benchmark and write the sketch to SSD. After the update benchmark, we flush all caches (page cache, directory entries, and inodes). We then map the sketch into RAM and perform estimate queries on the sketch.

Table 1 Size, width, and depth of the sketch and the number of elements inserted in count-min sketch and buffered count-min sketch in our benchmarks (update, estimate, and overestimate calculation).

Size	Width	Depth	#elements
128MB	3355444	5	9875188
256MB	6710887	5	19750377
512MB	13421773	5	39500754
1GB	26843546	5	79001508

This way we make sure that the sketch is not already cached in kernel caches from the update benchmark.

We compare the overestimates in the buffered count-min sketch and count-min sketch for all the four sketch sizes for which we perform update and estimate benchmarks. To measure the overestimates, we first perform the update benchmark. However, during the update benchmark, we also store each inserted element in a multiset. Once updates are done, we iterate over the multiset and query for the estimate of each element in the multiset. We then take the difference of the count returned from the sketch and the actual count of the element to calculate the overestimate.

For SSD-based experiments, we allocate space for the sketch by `mmap`-ing it to a file on SSD. We then control the available RAM to the benchmarking process using `cgroups`. We fix the RAM size for all the experiments to be $\approx$ 67MB. We then increase the size of the sketch based on the RAM-size-to-sketch-size ratio of the particular experiment. For the buffered count-min sketch, we use all the available RAM as the buffer. Paging is handled by the operating system based on the disk accesses. The point of these experiments is to evaluate the I/O efficiency of sketch operations.

All benchmarks were performed on a 64-bit Ubuntu 16.04 running Linux kernel 4.4.0-98-generic. The machine has Intel Skylake CPU U (Core(TM) i7-6700HQ CPU @ 2.60GHz with 4 cores and 6MB L3 cache) with 32 GB RAM and 1TB Toshiba SSD.

5.2 Configuring the sketch

In our benchmarks, we take as input δ, overestimate O $(= \epsilon n)$, and the size of the sketch S as configuration parameters. The depth of the sketch D is $\lceil \ln \frac{1}{\delta} \rceil$. The number of cells C is $S/CELL\_SIZE$. And width of the sketch is $\lceil e/\epsilon \rceil$.

Given these parameters, we calculate the number of elements n to be inserted in the sketch as $\frac{C \times O}{D \times e}$. In all our experiments, we fix δ to 0.01 and maximum overestimate to 8 and change the sketch size. Table 1 shows dimensions of the sketch and number of elements inserted based on the size of the sketch.

5.3 Update Performance

Figure 3 shows the update throughput of the count-min sketch and buffered count-min sketch with changing RAM-size-to-sketch-size ratios. The buffered count-min sketch is 3.7×–4.7× faster compared to the count-min sketch in terms of update throughput on SSD.

The buffered count-min sketch performs less than one I/O per update operation because all the hashes for a given element are localized to a single page on SSD. However, in the count-min sketch the hashes for a given element are spread across the whole sketch. Therefore,

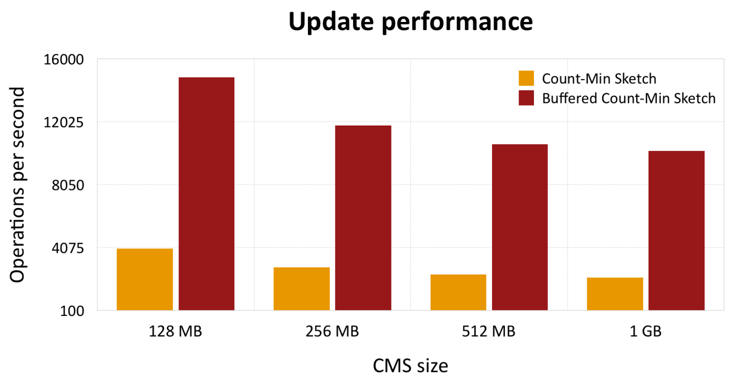


Figure 3 Update throughput of the count-min sketch and buffered count-min sketch with increasing sizes. The available RAM is fixed to ≈ 64MB. With increasing sketch sizes (on x-axis) the RAM-size-to-sketch-size is also increasing 2, 4, 8, and 16. (Higher is better.)

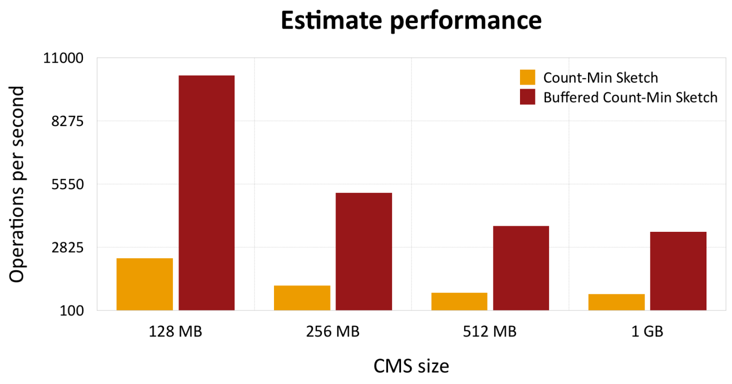


Figure 4 Estimate throughput of the count-min sketch and buffered count-min sketch with increasing sizes. The available RAM is fixed to ≈ 64MB. With increasing sketch sizes (on x-axis) the RAM-size-to-sketch-size is also increasing 2, 4, 8, and 16. (Higher is better.)

the update throughput of the buffered count-min sketch is 3.7× when the sketch is twice the size of the RAM. And the difference in the throughput increases as the sketch gets bigger and RAM size stays the same.

5.4 Estimate Performance

Figure 4 shows the estimate throughput of the count-min sketch and buffered count-min sketch with changing RAM-size-to-sketch-size ratios. The buffered count-min sketch is ≈ 4.3× faster compared to the count-min sketch in terms of estimate throughput on SSD.

The buffered count-min sketch performs a single I/O per estimate operation because all the hashes for a given element are localized to a single page on SSD. In comparison, count-min sketch may have to perform as many as h I/Os per estimate operation, where h is the depth of the count-min sketch.

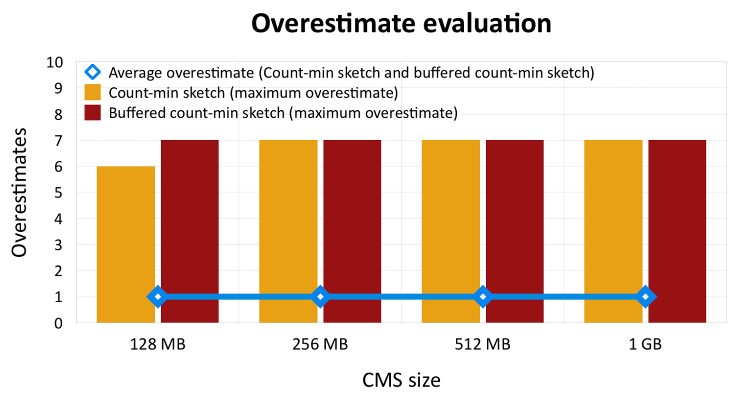


Figure 5 Maximum overestimate reported by the count-min sketch and buffered count-min sketch for any inserted element for different sketch sizes. The blue line represents the average overestimate reported by the count-min sketch and buffered count-min sketch for all the inserted elements. The average overestimate is same for both the count-min sketch and buffered count-min sketch.

5.5 Overestimates

In Figure 5 we empirically compare overestimates returned by the count-min sketch and buffered count-min sketch for all the four sketch sizes for which we performed update and estimate benchmarks. And we found that the average and the maximum overestimate returned from the count-min sketch and buffered count-min sketch are exactly the same. This shows that empirically hash localization in the buffered count-min sketch does not have any major effect on the overestimates.

6 Conclusion

In this paper we implemented and mathematically analyzed the buffered count-min sketch and empirically showed that our implementation achieves 3.7×–4.7× the speedup on update (insert) and 4.3× speedup on estimate (lookup) operations. Queries take 1 I/O, which is optimal in the worst case if not allowed to buffer. However, we do not know whether the update time is optimal. To the best of our knowledge, no lower bounds on the update time of such a data structure are known (the only known upper bounds are on space, e.g., in [15]). We leave the question of deriving update lower bounds and/or a SSD-based data structure with faster update time for future work.

References

1 Alok Aggarwal and S. Vitter, Jeffrey. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988. `doi:10.1145/48529.48535`.

2 Austin Appleby. 32-bit variant of murmurhash3, 2011. URL: `https://sites.google.com/site/murmurhash/`.

3 Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 1–16, New York, NY, USA, 2002. ACM. `doi:10.1145/543613.543615`.

4 Michael A. Bender, Martin Farach-Colton, Rob Johnson, Russell Kraner, Bradley C. Kuszmaul, Dzejla Medjedovic, Pablo Montes, Pradeep Shetty, Richard P. Spillane, and Erez

Zadok. Don't thrash: How to cache your hash on flash. *Proc. VLDB Endow.*, 5(11):1627–1637, jul 2012. doi:10.14778/2350229.2350275.

5 Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970. doi:10.1145/362686.362692.

6 Flavio Bonomi, Michael Mitzenmacher, Rina Panigrahy, Sushil Singh, and George Varghese. An improved construction for counting bloom filters. In *Proceedings of the 14th Conference on Annual European Symposium - Volume 14*, ESA'06, pages 684–695, London, UK, UK, 2006. Springer-Verlag. doi:10.1007/11841036_61.

7 Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 126–134, 1999.

8 Mustafa Canim, George A. Mihaila, Bishwaranjan Bhattacharjee, Christian A. Lang, and Kenneth A. Ross. Buffered bloom filters on solid state storage. In Rajesh Bordawekar and Christian A. Lang, editors, *ADMS@VLDB*, pages 1–8, 2010. URL: http://dblp.uni-trier.de/db/conf/vldb/adms2010.html#CanimMBLR10.

9 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP '02, pages 693–703, Berlin, Heidelberg, 2002. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=646255.684566.

10 Aiyou Chen, Yu Jin, Jin Cao, and Li Erran Li. Tracking long duration flows in network traffic. In *Proceedings of the 29th Conference on Information Communications*, INFOCOM'10, pages 206–210, Piscataway, NJ, USA, 2010. IEEE Press. URL: http://dl.acm.org/citation.cfm?id=1833515.1833557.

11 Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005. doi:10.1016/j.jalgor.2003.12.001.

12 Biplob Debnath, Sudipta Sengupta, Jin Li, David J. Lilja, and David H. C. Du. Bloom-flash: Bloom filter on flash-based storage. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*, ICDCS '11, pages 635–644, Washington, DC, USA, 2011. IEEE Computer Society. doi:10.1109/ICDCS.2011.44.

13 Ehsan Eydi, Dzejla Medjedovic, Emina Mekic, and Elmedin Selmanovic. Buffered count-min sketch. In Mirsad Hadžikadić and Samir Avdaković, editors, *Advanced Technologies, Systems, and Applications II*, pages 249–255, Cham, 2018. Springer International Publishing.

14 Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Mining data streams: A review. *SIGMOD Rec.*, 34(2):18–26, 2005. doi:10.1145/1083784.1083789.

15 Sumit Ganguly. Lower bounds on frequency estimation of data streams. In *International Computer Science Symposium in Russia*, pages 204–215. Springer, 2008.

16 Michael Gertz, Quinn Hart, Carlos Rueda, Shefali Singhal, and Jie Zhang. A data and query model for streaming geospatial image data. In Torsten Grust, Hagen Höpfner, Arantza Illarramendi, Stefan Jablonski, Marco Mesiti, Sascha Müller, Paula-Lavinia Patranjan, Kai-Uwe Sattler, Myra Spiliopoulou, and Jef Wijsen, editors, *Current Trends in Database Technology – EDBT 2006*, pages 687–699, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

17 Amit Goyal, Jagadeesh Jagarlamudi, Hal Daumé, III, and Suresh Venkatasubramanian. Sketch techniques for scaling distributional similarity to the web. In *Proceedings of the 2010 Workshop on GEometrical Models of Natural Language Semantics*, GEMS '10, pages 51–56, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL: http://dl.acm.org/citation.cfm?id=1870516.1870524.

18 Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*,

VLDB '02, pages 346–357. VLDB Endowment, 2002. URL: `http://dl.acm.org/citation.cfm?id=1287369.1287400`.

19 Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 250–262, New York, NY, USA, 2004. ACM. `doi:10.1145/1031495.1031525`.

20 Prashant Pandey, Michael A. Bender, Rob Johnson, and Robert Patro. A general-purpose counting filter: Making every bit count. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 775–787, 2017. `doi:10.1145/3035918.3035963`.

21 Martin Raab and Angelika Steger. "balls into bins"—a simple and tight analysis. *Randomization and Approximation Techniques in Computer Science*, pages 159–170, 1998.

22 Tim Roughgarden and Gregory Valiant. Cs168: The modern algorithmic toolbox lecture #2: Approximate heavy hitters and the count-min sketch, 2018.

23 Tamás Sarlós, András A. Benczúr, Károly Csalogány, Dániel Fogaras, and Balázs Rácz. To randomize or not to randomize: Space optimal summaries for hyperlink analysis. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 297–306, New York, NY, USA, 2006. ACM. `doi:10.1145/1135777.1135823`.

24 Stuart Schechter, Cormac Herley, and Michael Mitzenmacher. Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks. In *Proceedings of the 5th USENIX Conference on Hot Topics in Security*, HotSec'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association. URL: `http://dl.acm.org/citation.cfm?id=1924931.1924935`.

25 David P. Woodruff. New algorithms for heavy hitters in data streams. *CoRR*, abs/1603.01733, 2016. URL: `http://arxiv.org/abs/1603.01733`, `arXiv:1603.01733`.

26 Yin Zhang, Sumeet Singh, Subhabrata Sen, Nick Duffield, and Carsten Lund. Online identification of hierarchical heavy hitters: Algorithms, evaluation, and applications. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, pages 101–114, New York, NY, USA, 2004. ACM. `doi:10.1145/1028788.1028802`.

27 Qi (George) Zhao, Mitsunori Ogihara, Haixun Wang, and Jun (Jim) Xu. Finding global icebergs over distributed data sets. In *Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '06, pages 298–307, New York, NY, USA, 2006. ACM. `doi:10.1145/1142351.1142394`.

Scalable Katz Ranking Computation in Large Static and Dynamic Graphs

Alexander van der Grinten
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
avdgrinten@hu-berlin.de

Elisabetta Bergamini
Karlsruhe Institute of Technology (KIT), Germany

Oded Green
School of Computational Science and Engineering, Georgia Institute of Technology, USA
ogreen@gatech.edu

David A. Bader
School of Computational Science and Engineering, Georgia Institute of Technology, USA
bader@gatech.edu

Henning Meyerhenke
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
meyerhenke@hu-berlin.de

Abstract

Network analysis defines a number of centrality measures to identify the most central nodes in a network. Fast computation of those measures is a major challenge in algorithmic network analysis. Aside from closeness and betweenness, Katz centrality is one of the established centrality measures. In this paper, we consider the problem of computing rankings for Katz centrality. In particular, we propose upper and lower bounds on the Katz score of a given node. While previous approaches relied on numerical approximation or heuristics to compute Katz centrality rankings, we construct an algorithm that iteratively improves those upper and lower bounds until a correct Katz ranking is obtained. We extend our algorithm to dynamic graphs while maintaining its correctness guarantees. Experiments demonstrate that our static graph algorithm outperforms both numerical approaches and heuristics with speedups between 1.5× and 3.5×, depending on the desired quality guarantees. Our dynamic graph algorithm improves upon the static algorithm for update batches of less than 10000 edges. We provide efficient parallel CPU and GPU implementations of our algorithms that enable near real-time Katz centrality computation for graphs with hundreds of millions of nodes in fractions of seconds.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms, Theory of computation → Parallel algorithms

Keywords and phrases network analysis, Katz centrality, top-k ranking, dynamic graphs, parallel algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.42

Related Version A full version of the paper is available at [20], `https://arxiv.org/abs/1807.03847`.

Funding Most of the work was done while AvdG was affiliated with University of Cologne, Germany, and HM with Karlsruhe Institute of Technology and University of Cologne. Additionally, EB, AvdG and HM were partially supported by grant ME 3619/3-2 within German Research

© Alexander van der Grinten, Elisabetta Bergamini, Oded Green, David A. Bader, and Henning Meyerhenke;
licensed under Creative Commons License CC-BY

Foundation (DFG) Priority Programme 1736. Funding was also provided by Karlsruhe House of Young Scientists via the International Collaboration Package. Funding for OG and DB was provided in part by the Defense Advanced Research Projects Agency (DARPA) under Contract Number FA8750-17-C-0086. The content of the information in this document does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

1 Introduction

Finding the most important nodes of a network is a major task in network analysis. To this end, numerous centrality measures have been introduced in the literature. Examples of well-known measures are betweenness (which ranks nodes according to their participation in the shortest paths of the network) and closeness (which indicates the average shortest-path distance to other nodes). A major limitation of both measures is that they are based on the assumption that information flows through the networks following shortest paths only. However, this is often not the case in practice; think, for example, of traffic on street networks: it is easy to imagine reasons why drivers might prefer to take slightly longer paths. On the other hand, it is also quite unlikely that *much* longer paths will be taken.

Katz centrality [9] accounts for this by summing all walks starting from a node, but weighting them based on their length. More precisely, the weight of a walk of length i is α^i, where α is some attenuation factor smaller than 1. Thus, naming $\omega_i(v)$ the number of walks of length i starting from node v, the Katz centrality of v is defined as

$$\mathbf{c}(v) := \sum_{i=1}^{\infty} \omega_i(v)\,\alpha^i \tag{1}$$

or equivalently: $\mathbf{c} = \left(\sum_{i=1}^{\infty} A^i\,\alpha^i\right)\vec{I}$, where A is the adjacency matrix of the graph and $\vec{I}$ is the vector consisting only of 1s. This can be restated as a Neumann series, resulting in the closed-form expression $\mathbf{c} = \alpha A(I - \alpha A)^{-1}\vec{I}$, where I is the identity matrix. Thus, Katz centrality can be computed exactly by solving the linear system

$$(I - \alpha A)\,\mathbf{z} = \vec{I}\ , \tag{2}$$

followed by evaluating $\mathbf{c} = \alpha A\,\mathbf{z}$. We call this approach the *linear algebra formulation*. In practice, the solution to Eq. (2) is numerically approximated using iterative solvers for linear systems. While these solvers yield solutions of good quality, they can take hundreds of iterations to converge [17]. Thus, in terms of running time, those algorithms can be impractical for today's large networks, which often have millions of nodes and billions of edges.

Instead, Foster et al.'s [7] algorithm estimates Katz centrality iteratively by computing partial sums of the series from Eq. (1) until a stopping criterion is reached. Although very efficient in practice, this method has no guarantee on the correctness of the ranking it finds, not even for the top nodes. Thus, the approach is ineffective for applications where only a subset of the most central nodes is needed or when accuracy is needed. As this is indeed the case in many applications, several top-k centrality algorithms have been proposed recently for closeness [2] and betweenness [13]. Recently, a top-k algorithm for Katz centrality [17] was suggested. That algorithm still relies on solving Eq. (2); however, it reduces the numerical accuracy that is required to obtain a top-k rating. Similarly, Zhan et al. [21] propose a heuristic method to exclude certain nodes from top-k rankings but do not present algorithmic improvements on the actual Katz computation.

Dynamic graphs. Furthermore, many of today's real-world networks, such as social networks and web graphs, are dynamic in nature and some of them evolve over time at a very quick pace. For such networks, it is often impractical to recompute centralities from scratch after each graph modification. Thus, several dynamic graph algorithms that efficiently update centrality have been introduced for closeness [3] and betweenness [12]. Such algorithms usually work well in practice, because they reduce the computation to the part of the graph that has actually been affected. This offers potentially large speedups compared to recomputation. For Katz centrality, dynamic algorithms have recently been proposed by Nathan et al. [15, 16]. However, those algorithms rely on heuristics and are unable to reproduce the exact Katz ranking after dynamic updates.

Our contribution. We construct a vertex-centric algorithm that computes Katz centrality by iteratively improving upper and lower bounds on the centrality scores (see Section 3 for the construction of this algorithm). While the computed centralities are approximate, our algorithm guarantees the correct ranking. We extend (in Section 4) this algorithm to dynamic graphs while preserving the guarantees of the static algorithm. An extensive experimental evaluation (see Section 5) shows that (i) our new algorithm outperforms Katz algorithms that rely on numerical approximation with speedups between $1.5\times$ and $3.5\times$, depending on the desired correctness guarantees, (ii) our algorithm has a speedup in the same order of magnitude over the widely-used heuristic of Foster et al. [7] while improving accuracy, (iii) our dynamic graph algorithm improves upon static recomputation of Katz rankings for batch sizes of less than 10000 edges and (iv) efficient parallel CPU and GPU implementations of our algorithm allow near real-time computation of Katz centrality in fractions of seconds even for very large graphs. In particular, our GPU implementation achieves speedups of more than $10\times$ compared to a 20-core CPU implementation.

2 Preliminaries

2.1 Notation

Graphs. In the following sections, we assume that $G = (V, E)$ is the input graph to our algorithm. Unless stated otherwise, we assume that G is directed. For the purposes of Katz centrality, undirected graphs can be modeled by replacing each undirected edge with two directed edges in reverse directions. For a node $x \in V$, we denote the *out*-degree of x by $\deg(x)$. The maximum out-degree of any node in G is denoted by $\deg_{\max}$.

Katz centrality. The Katz centrality of the nodes of G is given by Eq. (1). With $\mathbf{c}_i(v)$ we denote the i-th partial sum of Eq. (1). Katz centrality is not defined for arbitrary values of α. In general, Eq. (1) converges for $\alpha < \frac{1}{\sigma_{\max}}$, where $\sigma_{\max}$ is the largest singular value of the adjacency matrix A (see [9]).

Katz centrality can also be defined by counting *inbound* walks in G [9, 18]. For this definition, $\omega_i(x)$ is replaced by the number of walks of length i that end in $x \in V$. Indeed, for applications like web graphs, nodes that are the target of many links intuitively should be considered more central than nodes that only have many links themselves[1]. However, as inbound Katz centrality coincides with the outbound Katz centrality of the reverse graph, we will not specifically consider it in this paper.

[1] This is a central idea behind the PageRank [5] metric.

2.2 Related work

Most algorithms that are able to compute Katz scores with approximation guarantees are based on the linear algebra formulation and compute a numerical solution to Eq. (2). Several approximation algorithms have been developed in order to decrease the practical running times of this formulation (e.g. based on low-rank approximation [1]). Nathan et al. [17] prove a relationship between the numerical approximation quality of Eq. (2) and the resulting Katz ranking quality. While this allows computation of top-k rankings with reduced numerical approximation quality, no significant speedups can be expected if full Katz rankings are desired.

Foster et al. [7] present a vertex-centric heuristic for Katz centrality: They propose to determine Katz centrality by computing the recurrence $c_{i+1} = \alpha A\ c_i + \vec{I}$. The computation is iterated until either a fixed point[2] or a predefined number of iterations is reached. This algorithm performs well in practice; however, due to the heuristic nature of the stopping condition, the algorithm does not give any correctness guarantees.

Another paper from Nathan et al. [16] discusses an algorithm for a "personalized" variant of Katz centrality. Our algorithm uses a similar iteration scheme but differs in multiple key properties of the algorithm: Instead of considering personalized Katz centrality, our algorithm computes the usual, "global" Katz centrality. While Nathan et al. give a global bound on the quality of their solution, we are able to compute per-node bounds that can guarantee the correctness of our ranking. Finally, Nathan et al.'s dynamic update procedure is a heuristic algorithm without correctness guarantee, although its ranking quality is good in practice. In contrast to that, our dynamic algorithm reproduces exactly the results of the static algorithm.

3 Iterative improvement of Katz bounds

3.1 Per-node bounds for Katz centrality

The idea behind our algorithm is to compute upper and lower bounds on the centrality of each node. Those bounds are iteratively improved. We stop the iteration once an application-specific stopping criterion is reached. When that happens, we say that the algorithm *converges*.

Per-node upper and lower bounds allow us to rank nodes against each other: Let $\ell_r(x)$ and $u_r(x)$ denote respectively lower and upper bounds on the Katz score of node x after iteration r. An explicit construction of those bounds will be given later in this section; for now, assume that such bounds exist. Furthermore, let w and v be two nodes; without loss of generality, we assume that w and v are chosen such that $\ell_r(w) \geq \ell_r(v)$. If $\ell_r(w) > u_r(v)$, then w appears in the Katz centrality ranking before v and we say that w and v are *separated* by the bounds ℓ_r and u_r. In this context, it should be noted that per-node bounds do not allow us to prove that the Katz scores of two nodes are equal[3]. However, as the algorithm still needs to be able to rank nodes x that share the same $\ell_r(x)$ and $u_r(x)$ values, we need a more relaxed concept of separation. Therefore:

▶ **Definition 1.** In the same setting as before, let $\epsilon > 0$. We say that w and v are ϵ*-separated*, if and only if

$$\ell_r(w) > u_r(v) - \epsilon\ . \tag{3}$$

[2] Note that a true fixed point will not be reached using this method unless the graph is a DAG.
[3] In theory, the linear algebra formulation is able to prove that the score of two nodes is indeed equal. However, in practice, limited floating point precision limits the usefulness of this property.

Intuitively, the introduction of ϵ makes the ϵ-condition easier to fulfill than the separation condition: Indeed, separated pairs of nodes are also ϵ-separated for every $\epsilon > 0$. In particular, ϵ-separation allows us to construct Katz rankings even in the presence of nodes that have the same Katz score: Those nodes are never separated, but they will eventually be ϵ-separated for every $\epsilon > 0$.

In order to actually construct rankings, it is sufficient to notice that once all pairs of nodes are ϵ-separated, sorting the nodes by their lower bounds ℓ_r yields a correct Katz ranking, except for pairs of nodes with a difference in Katz score of less than ϵ. Thus, using this definition, we can discuss possible stopping criteria for the algorithm:

Ranking criterion. Stop once all nodes are ϵ-separated from each other. This guarantees that the ranking is correct, except for nodes with scores that are very close to each other.

Top-k criterion. Stop once the top-k nodes are ϵ-separated from each other and from all other nodes. For $k = n$ this criterion reduces to the ranking criterion.

Score criterion. Stop once the difference between the upper and lower bound of each node becomes less than ϵ. This guarantees that the Katz centrality of each node is correct up to an additive constant of ϵ.

Pair criterion. Stop once two given nodes u and v are ϵ-separated.

First, we notice that a simple lower bound on the Katz centrality of a node v can be obtained by truncating the series in Eq. (1) after r iterations, hence, $\ell_r(v) := \sum_{i=1}^{r} \omega_i(v)\alpha^i$ is a lower bound on $\mathbf{c}(v)$. For undirected graphs, this lower bound can be improved to $\sum_{i=1}^{r} \omega_i(v)\alpha^i + \omega_r(v)\alpha^{r+1}$, as any walk of length r can be extended to a walk of length $r+1$ with the same starting point by repeating its last edge with reversed direction.

▶ **Theorem 2.** *Let $\gamma = \frac{\deg_{\max}}{1-\alpha \deg_{\max}}$. For any $r \geq 1$, $v \in V$ and $\alpha < \frac{1}{\deg_{\max}}$, the value*

$$u_r(v) := \sum_{i=1}^{r} \alpha^i \omega_i(v) + \alpha^{r+1}\omega_r(v)\gamma$$

is an upper bound on $\mathbf{c}(v)$.

Proof. First, let $S_i(v)$ be the set of nodes x for which there exists a walk of length i starting in v and ending in x. Each walk of length $i+1$ is the concatenation of a walk of length i ending in $x \in S_i(v)$ and an edge (x, y), where y is some neighbor of x. Let $\omega_i(v, x)$ denote the number of walks of length i that start in v and end in x. Thus, we can write

$$\omega_{i+1}(v) = \sum_{x \in S_i(v)} \deg(x)\ \omega_i(v, x) \leq \sum_{x \in S_i(v)} \deg_{\max}\ \omega_i(v, x) = \deg_{\max}\ \omega_i(v)\ . \tag{4}$$

By applying induction to the previous inequality, it is easy to see that, for any $j > 1$,

$$\omega_{i+j}(v) \leq (\deg_{\max})^j \omega_i(v)\ .$$

Discarding the first r terms of the sum in Eq. (1) then yields

$$\sum_{i=r+1}^{\infty} \alpha^i \omega_i(v) \leq \sum_{j=1}^{\infty} \alpha^{r+j} (\deg_{\max})^j \omega_r(v) = \alpha^r \omega_r(v) \sum_{j=1}^{\infty} (\alpha \deg_{\max})^j$$
$$= \alpha^r \omega_r(v) \left(\frac{1}{1 - \alpha \deg_{\max}} - 1 \right) = \alpha^{r+1} \omega_r(v)\gamma\ .$$

For the second to last equality, we rewrite the infinite series as a geometric sum. ◀

The following lemma (proof in the full version of this paper [20]) shows that we can indeed iteratively improve the upper and lower bounds for each node $x \in V$:

▶ **Lemma 3.** *For each $x \in V$, $\ell_i(x)$ is non-decreasing in i and $u_i(x)$ is non-increasing in i.*

Theorem 2 requires us to choose $\alpha < \frac{1}{\deg_{\max}}$, which is a restriction compared to the more general requirement of $\alpha < \frac{1}{\sigma_{\max}}$. For our experiments in the later sections of this paper, we set $\alpha = \frac{1}{1+\deg_{\max}}$ in order to satisfy this condition. Aside from enabling us to apply the theorem, this choice of α has some additional advantages: First, because Theorem 2 gives an upper bound on Eq. (1), Katz centrality is guaranteed to converge for this value of the α parameter[4]. $\deg_{\max}$ is also much easier to compute than $\sigma_{\max}$, an operation that is comparable in complexity to computing the Katz centrality itself[5]. Finally, $\alpha = \frac{1}{1+\deg_{\max}}$ is widely-used in existing literature [4, 7], with Foster et al. calling it the "generally-accepted default attenuation factor" [7].

It is worth remarking (proof in the full version of this paper [20]) that graphs exist for which the bound from Theorem 2 is sharp:

▶ **Lemma 4.** *If G is a complete graph, $u_i(x) = \mathbf{c}(x)$ for all $x \in V$ and $i \in \mathbb{N}$.*

3.2 Efficient rankings using per-node bounds

In the following, we state the description of our Katz algorithm for static graphs. As hinted earlier, the algorithm estimates Katz centrality by computing $u_r(v)$ and $\ell_r(v)$. These upper and lower bounds are iteratively improved by incrementing r until the algorithm converges.

To actually compute $\mathbf{c}_r(v)$, we use the well-known fact that the number of walks of length i starting in node v is equal to the sum of the number of walks of length $i-1$ starting in the neighbors of v, in other words:

$$\omega_i(v) = \sum_{v \to x \in E} \omega_{i-1}(x) \ . \tag{5}$$

Thus, if we initialize $\omega_1(v)$ to $\deg(v)$ for all $v \in V$, we can then repeatedly loop over the edges of G and compute tighter and tighter lower bounds.

We focus here on the top-k convergence criterion. It is not hard to see how our techniques can be adopted to the other stopping criteria mentioned at the start of the previous subsection. To be able to efficiently detect convergence, the algorithm maintains a set of *active* nodes. These are the nodes for which the lower and upper bounds have not yet converged. Initially, all nodes are active. Each node is *deactivated* once it is ϵ-separated from the k nodes with highest lower bounds ℓ_r. It should be noted that, because of Lemma 3, deactivated nodes will stay deactivated in all future iterations. Thus, for the top-k criterion, it is sufficient to check whether (i) only k nodes remain active and (ii) the remaining active nodes are ϵ-separated from each other. This means that each iteration will require less work than its previous iteration.

Algorithm 1 depicts the pseudocode of the algorithm. Computation of $\omega_r(v)$ is done by evaluating the recurrence from Eq. (5). After the algorithm terminates, the ϵ-separation property guarantees that the k nodes with highest $\ell_r(v)$ form a top-k Katz centrality ranking (although $\ell_r(v)$ does not necessarily equal the true Katz score).

[4] This was already noticed by Katz [9] and can alternatively be proven through linear algebra.
[5] Indeed, the popular power iteration method to compute $\sigma_{\max}$ for real, symmetric, positive-definite matrices has a complexity of $\Omega(r\,|E|)$, where r denotes a number of iterations.

Algorithm 1 Katz centrality bound computation for static graphs.

```
$\gamma \leftarrow \deg_{\max} / (1 - \alpha \deg_{\max})$
Initialize $\mathbf{c}_0(x) \leftarrow 0 \quad \forall x \in V$
Initialize $r \leftarrow 0$ and $\omega_0(x) \leftarrow 1 \quad \forall x \in V$
Initialize set of active nodes: $M \leftarrow V$
while not CONVERGED() do
    Set $r \leftarrow r + 1$ and $\omega_r(x) \leftarrow 0 \quad \forall x \in V$
    for all $v \in V$ do
        for all $v \to u \in E$ do
            $\omega_r(v) \leftarrow \omega_r(v) + \omega_{r-1}(u)$
        $\mathbf{c}_r(v) \leftarrow \mathbf{c}_{r-1}(v) + \alpha^r \omega_r(v)$
        if $G$ undirected then
            $\ell_r(v) \leftarrow \mathbf{c}_r(v) + \alpha^{r+1} \omega_r(v)$
        else
            $\ell_r(v) \leftarrow \mathbf{c}_r(v)$
        $u_r(v) \leftarrow \mathbf{c}_r(v) + \alpha^{r+1} \omega_r(v) \gamma$

function CONVERGED()
    PARTIALSORT($M$, $k$, $\ell_r$, decreasing)
    for all $i \in \{k+1, \ldots, |V|\}$ do
        if $u_r(M[i]) - \epsilon < \ell_r(M[k])$ then
            $M \leftarrow M \setminus \{v\}$
    if $|M| > k$ then
        return false
    for all $i \in \{2, \ldots, \min(|M|, k)\}$ do
        if $u_r(M[i]) - \epsilon \geq \ell_r(M[i-1])$ then
            return false
    return true
```

The CONVERGED procedure in Algorithm 1 checks whether the top-k convergence criterion is satisfied. In this procedure, M denotes the set of active nodes. The procedure first partially sorts the elements of M by decreasing lower bound ℓ_r. After that is done, the first k elements of M correspond to the top-k elements in the current ranking (which might not be correct yet). Note that it is not necessary to construct the entire ranking here; sorting just the top-k nodes is sufficient. The procedure tries to deactivate nodes that cannot be in the top-k and afterwards checks if the remaining top-k nodes are correctly ordered. These checks are performed by testing if the ϵ-separation condition from Eq. (3) is true.

Complexity analysis. The sequential worst-case time complexity of Algorithm 1 is $\mathcal{O}(r\,|E| + r\,\mathcal{C})$, where r is the number of iterations and $\mathcal{C}$ is the complexity of the convergence checking procedure. It is easy to see that the loop over V can be parallelized, yielding a complexity of $\mathcal{O}(r\,\frac{|V|}{p}\deg_{max} + r\,\mathcal{C})$ on a parallel machine with p processors. The complexity of CONVERGED, the top-k ranking convergence criterion, is dominated by the $\mathcal{O}(|V| + k\log k)$ complexity of partial sorting. Both the score and the pair criteria can be implemented in $\mathcal{O}(1)$.

It should be noted that – for the same solution quality – our algorithm converges at least as fast as the heuristic of Foster et al. that computes a Katz ranking without correctness guarantee. Indeed, the values of $\mathbf{c}_r$ yield exactly the values that are computed by the heuristic. However, Foster et al.'s heuristic is unable to accurately assess the quality of its current solution and might thus perform too many or too few iterations.

4 Updating Katz centrality in dynamic graphs

In this section, we discuss how our Katz centrality algorithm can be extended to compute Katz centrality rankings for dynamically changing graphs. We model those graphs as an initial graph that is modified by a sequence of edge insertions and edge deletions. We do not explicitly handle node insertions and deletions as those can easily be supported by adding enough isolated nodes to the initial graph.

Before processing any edge updates, we assume that our algorithm from Section 3 was first executed on the initial graph to initialize the values $\omega_i(x)$ for all $x \in V$. The dynamic graph algorithm needs to recompute $\omega_i(x)$ for $i \in \{1, \ldots, r\}$, where r is the number of iterations that was reached by the static Katz algorithm on the initial graph. The main observation here is that if an edge $u \to v$ is inserted into (or deleted from) the initial graph, $\omega_i(x)$ only changes for nodes x in the vicinity of u. More precisely, $\omega_i(x)$ can only change if u is reachable from x in at most $i - 1$ steps.

Algorithm 2 Dynamic Katz update procedure.

```
$E \leftarrow E \setminus \mathcal{D}$
$S \leftarrow \emptyset$, $T \leftarrow \emptyset$
for all $w \to v \in \mathcal{I} \cup \mathcal{D}$ do
    $S \leftarrow S \cup \{w\}$
    $T \leftarrow T \cup \{v\}$
for all $i \in \{1, \ldots, r\}$ do
    updateLevel(i)
for all $w \in S$ do
    Recompute $\ell_r(w)$ and $u_r(w)$ from $\mathbf{c}_r(w)$
for all $w \in V$ do
    if $u_r(w) \geq \min_{x \in M} \ell_r(x) - \epsilon$ then
        $M \leftarrow M \cup \{w\}$                ▷ Reactivation
$E \leftarrow E \cup \mathcal{I}$
while not converged() do
    Run more iterations of static algorithm

procedure updateLevel(i)
    for all $v \in S \cup T$ do
        $\omega'_i(v) \leftarrow \omega_i(v)$
    for all $v \in S$ do
        for all $w \to v \in E$ do
            $S \leftarrow S \cup \{w\}$
            $\omega'_i(w) \leftarrow \omega'_i(w) - \omega_{i-1}(v) + \omega'_{i-1}(v)$
    for all $w \to v \in \mathcal{I}$ do
        $\omega'_i(w) \leftarrow \omega'_i(w) + \omega'_{i-1}(v)$
    for all $w \to v \in \mathcal{D}$ do
        $\omega'_i(w) \leftarrow \omega'_i(w) - \omega_{i-1}(v)$
    for all $w \in S$ do
        $\mathbf{c}_i(w) \leftarrow \mathbf{c}_i(w) - \alpha^i \omega_i(w) + \alpha^i \omega'_i(w)$
```

Algorithm 2 depicts the pseudocode of our dynamic Katz algorithm. $\mathcal{I}$ denotes the set of edges to be inserted, while $\mathcal{D}$ denotes the set of edges to be deleted. We assume that $\mathcal{I} \cap E = \emptyset$ and $\mathcal{D} \subseteq E$ before the algorithm. Effectively, the algorithm performs a breadth-first search (BFS) through the reverse graph of G and updates ω_i for all nodes nodes that were reached in steps 1 to i.

After the update procedure terminates, the new upper and lower bounds can be computed from $\mathbf{c}_r$ as in the static algorithm. We note that $\omega'_i(x)$ matches exactly the value of $\omega_i(x)$ that the static Katz algorithm would compute for the modified graph. Hence, the dynamic algorithm reproduces the correct values of $\mathbf{c}_r(x)$ and also of $\ell_r(x)$ and $u_r(x)$ for all $x \in V$. In case of the top-k convergence criterion, some nodes might need to be *reactivated* afterwards: Remember that the top-k criterion maintains a set M of active nodes. After edge updates are processed, it can happen that there are nodes x that are not ϵ-separated from all nodes in M anymore. Such nodes x need to be added to M in order to obtain a correct ranking. The ranking itself can then be updated by sorting M according to decreasing ℓ_r.

It should be noted that there is another related corner case: Depending on the convergence criterion, it can happen that the algorithm is not converged anymore even after nodes have been reactivated. For example, for the top-k criterion, this is the case if the nodes in M are not ϵ-separated from each other anymore. Thus, after the dynamic update we have to perform a convergence check and potentially run additional iterations of the static algorithm until it converges again.

Assuming that no further iterations of the static algorithms are necessary, the complexity of the update procedure is $\mathcal{O}(r\,|E| + \mathcal{C})$, where $\mathcal{C}$ is the complexity of convergence checking (see Section 3). In reality, however, the procedure can be expected to perform much better: Especially for the first few iterations, we expect the set S of vertices visited by the BFS to be much smaller than $|V|$. However, this implies that effective parallelization of the dynamic graph algorithm is more challenging than the static counterpart. We mitigate this problem, by aborting the BFS if $|S|$ becomes large and just update the ω_i scores unconditionally for all nodes.

Finally, it is easy to see that the algorithm can be modified to update ω in-place instead of constructing a new ω' matrix. For this optimization, the algorithm needs to save the value of ω_i for all nodes of S before overwriting it, as this value is required for iteration $i + 1$. For readability, we omit this modification in the pseudocode.

Table 1 Performance of the Katz algorithm, ranking criterion.

ϵ	r[a)]	Runtime[a)]	Separation[b)]	ϵ	r[a)]	Runtime[a)]	Separation[b)]
10^{-1}	2.3	33.51 s	96.189974 %	10^{-7}	7.2	78.74 s	99.994959 %
10^{-2}	3.0	42.81 s	98.478250 %	10^{-8}	7.9	83.28 s	99.998866 %
10^{-3}	3.8	51.59 s	99.264726 %	10^{-9}	8.6	85.10 s	99.998886 %
10^{-4}	4.8	65.99 s	99.391884 %	10^{-10}	9.2	89.03 s	99.998889 %
10^{-5}	5.7	71.53 s	99.992908 %	10^{-11}	9.8	99.43 s	99.998934 %
10^{-6}	6.5	70.59 s	99.994861 %	10^{-12}	10.4	96.86 s	99.998934 %
Foster	11.2	105.03 s	-	CG	12.0	117.24 s	-

a) Average over all instances. r is the number of iterations.
b) Fraction of node pairs that are separated (and not only ϵ-separated). Lower bound on the correctly ranked pairs. This is the geometric mean over all graphs.

5 Experiments

Implementation details. The new algorithm in this paper is hardware independent and as such we can implement it on different types of hardware with the right type of software support. Specifically, our dynamic Katz centrality requires a dynamic graph data structure. On the CPU we use NetworKit [19]; on the GPU we use Hornet[6]. The Hornet data structure is architecture independent, though at time of writing only a GPU implementation exists.

NetworKit consists of an optimized C++ network analysis library and bindings to access this library from Python. NetworKit contains parallel shared-memory implementations of many popular graph algorithms and can handle networks with billions of edges.

The Hornet [6], an efficient extension to the cuSTINGER [8] data structure, is a dynamic graph and matrix data structure designed for large scale networks and to support graphs with trillions of vertices. In contrast to cuSTINGER, Hornet better utilizes memory, supports memory reclamation, and can be updated almost ten times faster.

In our experiments, we compare our new algorithm to Foster et al.'s heuristic and a conjugate gradient (CG) algorithm (without preconditioning) that solves Eq. (2). The performance of CG could be possibly improved by employing a suitable preconditioner; however, we do not expect this to change our results qualitatively. Both of these algorithms were implemented in NetworKit and share the graph data structure with our new Katz implementation. We remark that for the static case, both CG and our Katz algorithm could be implemented on top of a CSR matrix data structure to improve the data locality and speed up the implementation.

Experimental setup. We evaluate our algorithms on a set of complex networks. The networks originate from diverse real-world applications and were taken from SNAP [14] and KONECT [11]. Details about the exact instances that we used can be found in the full version of this paper [20]. In order to be able to compare our algorithm to the CG algorithm, we turn the directed graphs in this test set into undirected graphs by ignoring edge directions. This ensures that the adjacency matrix is symmetric and CG is applicable. Our new algorithm itself would be able to handle directed graphs just fine.

[6] Hornet can be found at `https://github.com/hornet-gt`, while NetworKit is available from `https://github.com/kit-parco/networkit`. Both projects are open source, including the implementations of our new algorithm.

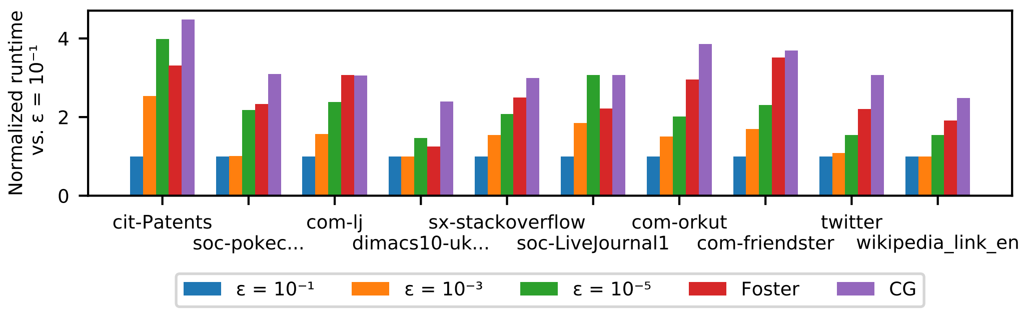


Figure 1 Katz performance on individual instances.

All CPU experiments ran on a machine with dual-socket Intel Xeon E5-2690 v2 CPUs with 10 cores per socket[7] and 128 GiB RAM. Our GPU experiments are conducted on an NVIDIA P100 GPU which has 56 Streaming Multiprocessors (SMs) and 64 Streaming Processors (SPs) per SM (for a total of 3584 SPs) and has 16GB of HBM2 memory. To effectively use the GPU, the number of active threads need to be roughly 8 times larger than the number of SPs. The Hornet framework has an API that enables such parallelization (with load balancing) such that the user only needs to write a few lines of code.

5.1 Evaluation of the static Katz algorithm

In a first experiment, we evaluate the running time of our static Katz algorithm. In particular, we compare it to the running time of the linear algebra formulation (i.e. the CG algorithm) and Foster et al.'s heuristic. We run CG until the residual is less than 10^{-15} to obtain a nearly exact Katz ranking (i.e. up to machine precision; later in this section, we compare to CG runs with larger error tolerances). For Foster's heuristic, we use an error tolerance of 10^{-9}, which also yields an almost exact ranking. For our own algorithm, we use the ranking convergence criterion (see Section 3) and report running times and the quality of our correctness guarantees for different values of ϵ. All algorithms in this experiment ran in single-threaded mode.

Table 1 summarizes the results of the evaluation. The fourth column of Table 1 states the fraction of separated pairs of nodes. This value represents a lower bound on the correctness of ranking. Note that pairs of nodes that have the same Katz score will never be separated. Indeed, this seems to be the case for about 0.001% of all pairs of nodes (as they are never separated, not even if ϵ is very low). Taking this into account, we can see that our algorithm already computes the correct ranking for 99% of all pairs of nodes at $\epsilon = 10^{-3}$. At this ϵ, our algorithm outperforms the other Katz algorithms considerably.

Furthermore, Table 1 shows that the average running time of our algorithm is smaller than the running time of the Foster et al. and CG algorithms. However, the graphs in our instance set vastly differ in size and originate from different applications; thus, the average running time alone does not give good indication for performance on individual graphs. In Figure 1 we report running times of our algorithm for the ten largest individual instances. $\epsilon = 10^{-1}$ is taken as baseline and the running times of all other algorithms are reported relative to this baseline. In the $\epsilon \leq 10^{-3}$ setups, our Katz algorithm outperforms the CG and

[7] Hyperthreading was disabled for the experiments.

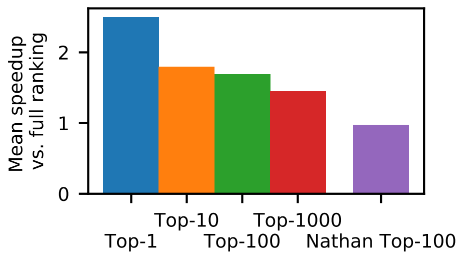


Figure 2 Top-k speedup over full ranking.

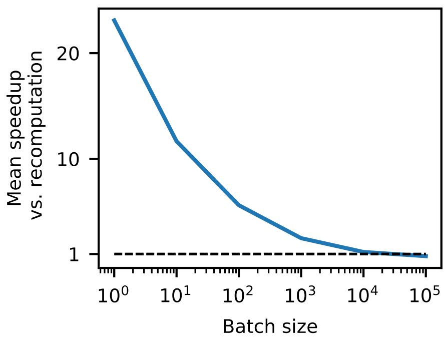


Figure 3 Dynamic update performance.

Foster et al. algorithms on all instances. Foster et al.'s algorithm is faster than our algorithm for $\epsilon = 10^{-5}$ on three out of ten instances. On the depicted instances, CG is never faster than our algorithm, although it can outperform our algorithm on some small instances and for very low ϵ.

Finally, in Figure 2, we present results of our Katz algorithm while using the top-k convergence criterion. We report (geometric) mean speedups relative to the full ranking criterion. The figure also includes the approach of Nathan et al. [17]. Nathan et al. conducted experiments on real-world graphs and concluded that solving Eq. (2) with an error tolerance of 10^{-4} in practice almost always results in the correct top-100 ranking. Thus, we run CG with that error tolerance. However, it turns out that this approach is barely faster than our full ranking algorithm. In contrast to that, our top-k algorithm yields decent speedups for $k \leq 1000$.

5.2 Evaluation of the dynamic Katz algorithm

In our next experiment, we evaluate the performance of our dynamic Katz algorithm to compute top-1000 rankings using $\epsilon = 10^{-4}$. We select b random edges from the graph, delete them in a single batch and run our dynamic update algorithm on the resulting graph. We vary the batch size b from 10^0 to 10^5 and report the running times of the dynamic graph algorithm relative to recomputation. Similar to the previous experiment, we run the algorithms in single-threaded mode. Note that while we only show results for edge deletion, edge insertion is completely symmetric in Algorithm 2.

Figure 3 summarizes the results of the experiment. For batch sizes $b \leq 1000$, our dynamic algorithm offers a considerable speedup over recomputation of Katz centralities. As many of the graphs in our set of instances have a small diameter, for larger batch sizes ($b > 10000$), almost all of the vertices of the graph need to be visited during the dynamic update procedure. Hence, the dynamic update algorithm is slower than recomputation in these cases.

5.3 Real-time Katz computation using parallel CPU and GPU implementations

Our last experiment concerns the practical running time and scalability of efficient parallel CPU and GPU implementations of our algorithm. For this, we compare the running times of our shared-memory CPU implementation with different numbers of cores. Furthermore, we report results of our GPU implementation. Because of GPU memory constraints, we could

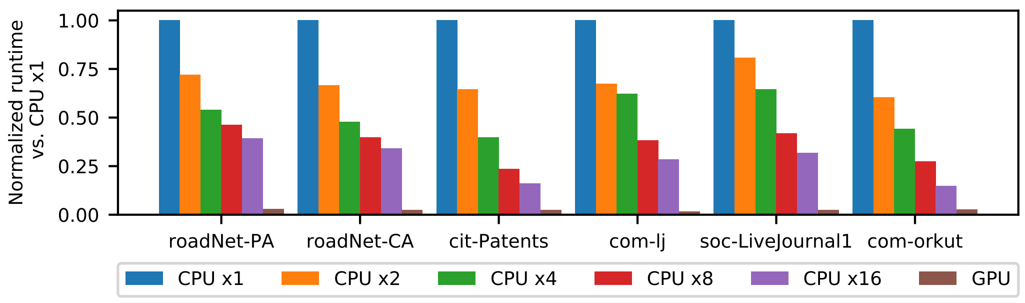


Figure 4 Scalability of parallel CPU and GPU implementations.

not process all of the graphs on the GPU. Hence, we provide the results of this experiment only for a subset of graphs that do fit into the memory of our GPU. The graphs in this subset have between 1.5 million and 120 million edges. We use the top-10000 convergence criterion with $\epsilon = 10^{-6}$.

Figure 4 depicts the results of the evaluation. In this figure, we consider the sequential CPU implementation as a baseline. We report the relative running times of the 2, 4, 8 and 16 core CPU configurations, as well as the GPU configuration, to this baseline. While the parallel CPU configurations yield moderate speedups over the sequential implementation, the GPU gives a significant speedup over the 16 core CPU configuration[8]. Even compared to a 20 core CPU configuration (not depicted in the plots; see the full version of this paper [20]), the GPU achieves a (geometric) mean speedup of $10\times$.

The CPU implementation achieves running times in the range of seconds; however, our GPU implementation reduces this running time to a fraction of a second. In particular, the GPU running time varies between 20 ms (for roadNet-PA) and 213 ms (for com-orkut), enabling near real-time computation of Katz centrality even for graphs with hundreds of millions of edges.

6 Conclusion

In this paper, we have presented an algorithm for Katz centrality that computes upper and lower bounds on the Katz score of individual nodes. Experiments demonstrated that our algorithm outperforms both linear algebra formulations and approximation algorithms, with speedups between 150% and 350% depending on desired correctness guarantees.

Future work could try to provide stricter per-node bounds for Katz centrality to further decrease the number of iterations that the algorithm requires to convergence. In particular, it would be desirable to prove per-node bounds that do not rely on $\alpha < 1/\deg_{\max}$. On the implementation side, our new algorithm could be formulated in the language of GraphBLAS [10] to enable it to run on a variety of upcoming software and hardware architectures.

[8] At time of writing, our CPU implementation uses a sequential algorithm for partial sorting; this is a bottleneck in the parallel CPU configurations.

References

1 E. Acar, D. M. Dunlavy, and T. G. Kolda. Link prediction on evolving data using matrix and tensor factorizations. In *2009 IEEE International Conference on Data Mining Workshops*, pages 262–269, Dec 2009. `doi:10.1109/ICDMW.2009.54`.

2 Elisabetta Bergamini, Michele Borassi, Pierluigi Crescenzi, Andrea Marino, and Henning Meyerhenke. Computing top-k closeness centrality faster in unweighted graphs. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 68–80. Society for Industrial and Applied Mathematics, 2018. `doi:10.1137/1.9781611974317.6`.

3 Patrick Bisenius, Elisabetta Bergamin, Eugenio Angriman, and Henning Meyerhenke. Computing top-k closeness centrality in fully-dynamic graphs. In *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 21–35. Society for Industrial and Applied Mathematics, 2018. `doi:10.1137/1.9781611975055.3`.

4 Francesco Bonchi, Pooya Esfandiar, David Gleich, Chen Greif, and Laks Lakshmanan. Fast matrix computations for pairwise and columnwise commute times and katz scores. *Internet Mathematics*, 8:73–112, 03 2012.

5 Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1):107–117, 1998. Proceedings of the Seventh International World Wide Web Conference. `doi:10.1016/S0169-7552(98)00110-X`.

6 F. Busato, O. Green, N. Bombieri, and D.A. Bader. Hornet: An Efficient Data Structure for Dynamic Sparse Graphs and Matrices on GPUs. In *IEEE Proc. High Performance Extreme Computing (HPEC)*, Waltham, MA, 2018.

7 Kurt C. Foster, Stephen Q. Muth, John J. Potterat, and Richard B. Rothenberg. A faster katz status score algorithm. *Computational & Mathematical Organization Theory*, 7(4):275–285, Dec 2001. `doi:10.1023/A:1013470632383`.

8 O. Green and D.A. Bader. cuSTINGER: Supporting Dynamic Graph Algorithms for GPUs. In *IEEE Proc. High Performance Embedded Computing Workshop (HPEC)*, Waltham, MA, 2016.

9 Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, Mar 1953. `doi:10.1007/BF02289026`.

10 J. Kepner, P. Aaltonen, D. Bader, A. Buluç, F. Franchetti, J. Gilbert, D. Hutchison, M. Kumar, A. Lumsdaine, H. Meyerhenke, S. McMillan, C. Yang, J. D. Owens, M. Zalewski, T. Mattson, and J. Moreira. Mathematical foundations of the graphblas. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9, Sept 2016. `doi:10.1109/HPEC.2016.7761646`.

11 Jérôme Kunegis. Konect: The koblenz network collection. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13 Companion, pages 1343–1350, New York, NY, USA, 2013. ACM. `doi:10.1145/2487788.2488173`.

12 Min-Joong Lee, Sunghee Choi, and Chin-Wan Chung. Efficient algorithms for updating betweenness centrality in fully dynamic graphs. *Information Sciences*, 326:278–296, 2016. `doi:10.1016/j.ins.2015.07.053`.

13 Min-Joong Lee and Chin-Wan Chung. Finding k-highest betweenness centrality vertices in graphs. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, pages 339–340, New York, NY, USA, 2014. ACM. `doi:10.1145/2567948.2577358`.

14 Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.

15 Eisha Nathan and David A. Bader. A dynamic algorithm for updating katz centrality in graphs. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in*

Social Networks Analysis and Mining 2017, ASONAM '17, pages 149–154, New York, NY, USA, 2017. ACM. doi:10.1145/3110025.3110034.

16 Eisha Nathan and David A. Bader. Approximating personalized katz centrality in dynamic graphs. In Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, and Konrad Karczewski, editors, *Parallel Processing and Applied Mathematics*, pages 290–302, Cham, 2018. Springer International Publishing.

17 Eisha Nathan, Geoffrey Sanders, James Fairbanks, Van Emden Henson, and David A. Bader. Graph ranking guarantees for numerical approximations to katz centrality. *Procedia Computer Science*, 108:68–78, 2017. International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland. doi:10.1016/j.procs.2017.05.021.

18 Mark Newman. *Networks: An Introduction.* Oxford University Press, Inc., New York, NY, USA, 2010.

19 Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. NetworKit: A tool suite for large-scale complex network analysis. *Network Science*, 4(4):508–530, 2016. doi:10.1017/nws.2016.20.

20 A. van der Grinten, E. Bergamini, O. Green, D.A. Bader, and Henning Meyerhenke. Scalable Katz ranking computation in large static and dynamic graphs. *arXiv*, 2018. arXiv:1807.03847.

21 Justin Zhan, Sweta Gurung, and Sai Phani Krishna Parsa. Identification of top-k nodes in large networks using katz centrality. *Journal of Big Data*, 4(1):16, May 2017. doi:10.1186/s40537-017-0076-5.

Round-Hashing for Data Storage: Distributed Servers and External-Memory Tables

Roberto Grossi
Dipartimento di Informatica, Università di Pisa, Italy
grossi@di.unipi.it

Luca Versari
Dipartimento di Informatica, Università di Pisa, Italy
luca.versari@di.unipi.it

Abstract

This paper proposes round-hashing, which is suitable for data storage on distributed servers and for implementing external-memory tables in which each lookup retrieves at most one single block of external memory, using a stash. For data storage, round-hashing is like consistent hashing as it avoids a full rehashing of the keys when new servers are added. Experiments show that the speed to serve requests is tenfold or more than the state of the art. In distributed data storage, this guarantees better throughput for serving requests and, moreover, greatly reduces decision times for which data should move to new servers as rescanning data is much faster.

2012 ACM Subject Classification Theory of computation → Sorting and searching, Information systems → Data dictionaries

Keywords and phrases consistent hashing, external memory, hash tables

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.43

Acknowledgements Work supported by MIUR. We thank Rasmus Pagh for pointing us the reference on AC^0-RAM dictionaries, and the anonymous referees for their comments.

1 Introduction

We study the problem of consistent hashing for data storage, where the keys are web pages or data items to be dynamically mapped to a set of m buckets, uniquely identified by integers in $[0 \dots m-1]$. At any time, we want to support the following operations (including `init()` for the initialization) to increase or decrease the number of buckets (i.e. change mapping).

- `numBuckets()`: Return the current number m of buckets.
- `findBucket(`u`)`: Given a key u, find its corresponding bucket identifier in $[0 \dots m-1]$.
- `newBucket()`: Add a new bucket having identifier m, thus making the range $[0 \dots m]$, and return the identifiers of the buckets whose keys should be redistributed.
- `freeBucket()`: Release the last bucket $m-1$, thus making range $[0 \dots m-2]$, and return the identifiers of the buckets whose keys should be redistributed.

Armed with the above operations, we can implement hashing by storing the keys in the buckets indicated by `findBucket()`, deciding when it is necessary to increase or decrease the number of buckets provided by `numBuckets()`: in the latter case, we have to redistribute the keys in the buckets indicated by `newBucket()` and `freeBucket()`.

© Roberto Grossi and Luca Versari;
licensed under Creative Commons License CC-BY

Table 1 Performance of the hashing methods for m buckets (servers). Here $s_0 \ll m$ is a constant slack parameter (typically $s = 64$ or 128), and $\alpha =$ (number of stored keys) $/\, m$ is the load factor. Although creating a new bucket moves $O(\alpha)$ keys on the average, each hashing method can take different time to decide which keys should be moved: "local" means that few other buckets scan their keys, while "distributed" means that all buckets scan their keys in parallel to decide which ones have to move to the new bucket. The $\tilde{O}()$ notation indicates an expected cost.

	find bucket	space	new bucket	notes
consistent hashing [13]	$O(\log m)$	$O(m)$	$\tilde{O}(\alpha + \log m)$	local
rendezvous hashing [24]	$O(\log m)$	$O(m)$	$O(m\alpha)$	distributed
jump consistent hash [14]	$\tilde{O}(\log m)$	$O(1)$	$O(m\alpha)$	distributed
linear hashing [15, 16]	$O(\log(m/s_0))$	$O(1)$	$\tilde{O}(s_0\alpha)$	local
round-hashing (ours)	$O(1)$	$O(1)$	$\tilde{O}(s_0\alpha)$	local, no division

History and motivation

Consistent hashing was invented by Karger et al. [13] for shared web caching, and highest random weight hashing (also known as rendezvous hashing) was invented by Thaler and Ravishankar [24] for web proxy servers. Both hashing methods were conceived independently around the mid 90s, and shared similar goals with different implementations: cached web pages are assigned to servers, so that when a server goes down, its cached web pages are reassigned to the other servers so as to preserve their load balancing; similarly, when a new server is added, some cached web pages are moved to it from the others. In contrast, a classical randomized load balancing scheme that uses hash mapping with modular operations on m is more expensive, as it requires to reallocate most of the keys when m changes.

Consistent hashing, in its basic version, maps both web pages and servers to the circular universe $[0 \ldots 2^w - 1]$, where each hash value requires w bits: each web page starts from its hash value in the circular universe and is assigned to the server whose hash value is clockwise met first; this can be done in $O(\log m)$ time using a search data structure of size $O(m)$ for m servers. Rendezvous hashing, for a given web page p, applies hashing to the pairs $\langle p, i\rangle$ for each server i, and then assigns p to the server $i = i_0$ that gives the maximum hash value among these pairs; this is computed in $O(\log m)$ time using a tree of size $O(m)$ as discussed by Wang and Ravishankar [26]. The first two rows of Table 1 report a summary of these bounds. Both methods apply their rule above when a server is deleted or added. They have been successfully exploited in the industry, e.g. Akamai, Microsoft's CARP, Chord [23], and Amazon's Dynamo [8] to name a few.

Recently, Lamping and Veach presented jump consistent hashing [14] at Google, observing that it can be tailored for data centers and data storage applications in general. In this scenario, servers cannot disappear, as this would mean loss of valuable data; rather, they can be added to increase storage capacity.[1] As a result, the hash values "jump" to higher values for the keys moved to a new bucket; moreover, the hash values are a contiguous range $[0 \ldots m - 1]$ for m servers, rather than a subset of m integers from $[0 \ldots 2^w - 1]$. This has a dramatic impact on the performance of the jump consistent hash, as illustrated in [14], observing that only balance and monotonicity should be guaranteed from the original proposal in [13]. The auxiliary storage is just $O(1)$, as shown in the third row of Table 1; average query cost is the m-th harmonic number, so $O(\log m)$, with no worst case guarantee.

[1] Data is split into shards, where each shard is handled by a cluster of machines with replication, thus it is not acceptable for shards to disappear [14].

We observe that linear hashing, introduced by Litwin [16] and Larson [15] at the beginning of the 80s, can also be successfully employed in this scenario: as reported in the fourth row of Table 1, the resulting cost is $O(\log m)$ time with $O(1)$ space, where $s_0 \leq m$ is a user-selectable parameter that can be conveniently fixed to be $s_0 = O(1)$.

Looking at the first four rows in Table 1, when a new bucket is created, $O(\alpha)$ keys on average are moved from the other buckets, where α is the load factor, namely, the number of stored keys divided by the number m of buckets.[2] However, the hashing methods take different time to decide which keys should be move. Specifically, consistent hashing has to examine the $O(\alpha)$ keys in the two neighbor servers in the worst case, and update the data structure in $O(\log m)$ time.[3] Rendezvous hashing requires that each bucket scans its keys and test whether the new bucket is now the maximum for some of them. Hence all the keys are scanned, $O(m\alpha)$, but only $O(\alpha)$ of them are moved in total. Jump consistent hashing needs to perform a similar task, to see which keys "jump" to the new bucket. Linear hashing requires to scan the keys in $s_0 = O(1)$ buckets to find the $O(\alpha)$ ones to move.

Our hashing scheme

In the scenario of consistent hashing for data storage, we present a new mapping scheme, called *round-mapping*, to implement the operations `init`(), `numBuckets`(), `findBucket`(u), `newBucket`(), and `freeBucket`() mentioned before. Based on this, we obtain *round-hashing*, which computes the hash value of the given key and invokes round-mapping for this value achieving $O(1)$ time and space in the worst case, as shown in the last row of Table 1. This is a desirable feature, as otherwise hashing with no worst-case guarantee can pose security threats, such as algorithmic complexity attacks [3, 7] for low-bandwidth denial of service exploiting its worst-case behavior. Our scheme adds new buckets in a round-robin fashion by interleaving them with the existing buffers, so as to grow stepwise. For a constant slack parameter $s_0 \ll m$ (typically $s_0 = 64$ or 128), round-hashing can guarantee that the number of keys in the most populated bucket is at most $1 + 1/s_0$ times the number of keys in the least populated bucket.

Compared to the other schemes in Table 1, round-hashing is much simpler and faster due to the fixed arithmetic scheme of round-mapping that avoids division. This brings us in the realm of the cost of instructions on a commodity processor. To concretely illustrate our points, we refer to Intel processors [11]. Here Euclidean division is not our friend: integer division and modulo operations on 64-bit integers take 85–100 cycles, whereas addition takes 1 cycle (and can be easily pipelined). Interestingly, this goes in the direction of the so-called AC^0-RAM dictionaries (e.g. see Andersson et al. [2]) and Practical RAM (e.g. see Brodnik et al. [4] and Miltersen [18]), where integer division and multiplication are not permitted, among others. However, multiplication should be taken with a grain of salt as, surprisingly, it takes 3–4 cycles (which becomes 1 cycle when it can be pipelined). Also, the modulo operation for powers of two or for small constants proportional to s_0, can be replaced with a few shift and multiplication operations [10] as available, for instance, in the `gcc` compiler from version 2.6. Our implementation of round-hashing avoids general integer division and modulo operations because they are almost two orders of magnitude slower than the other operations: using them could nullify the advantage of the $O(1)$ time complexity. Furthermore, adding buckets is also fast and a straight-forward modification of the scheme.

[2] We are assuming, wlog, that the buckets have all the same size.

[3] For the sake of discussion, we consider the basic version of consistent hashing, and refer the reader to [13, 14] for the version with multiple hash values per server.

Distributed servers

Motivated by the application to distributed servers, we performed an experimental study of the above hashing methods, applying our tuning wherever possible. The code is publicly available at `https://github.com/veluca93/round_hashing` to replicate the experiments.

Our first observation addresses how balanced are the buckets filled with the hashing methods in Table 1. By uniformly sampling all the possible keys, their hash values can be used to estimate how far the number of keys in buckets are from the ideal load factor α, reporting the least and the most populated buckets after the experiments. We observed that jump consistent hashing is very close to α, ranging from $0.988\,\alpha$ to $1.012\,\alpha$; the experimental study in [14] shows that it compares favorably with consistent hashing (rendezvous hashing is not directly compared). We can match this performance by setting $s_0 = 128$ for linear hashing and $s_0 = 64$ for round-hashing.

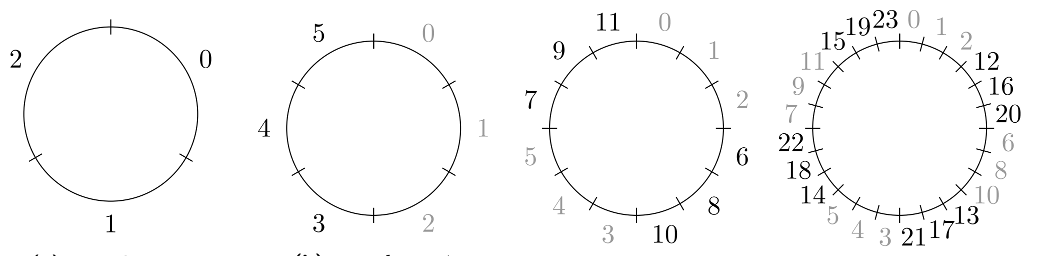
As a result of our tuning, to find the bucket number for a key, round-hashing is almost an order of magnitude faster than jump consistent hashing, and even much faster than the other hashing methods in Table 1. This is crucial for the system throughput: first, round-hashing can *serve* tenfold or more requests; second, when a new bucket number is added, it improves the performance of *rescanning* the keys to decide which ones move to the new bucket. We refer the reader to Section 3 for further details on our experimental study.

External-memory tables

It is interesting to apply round-hashing to high-throughput servers with many lookup requests, relatively few updates, and where some keys can be kept in a `stash` in main memory. We obtain a variant of dynamic hash tables, called *round-table*, and adopt the the EM model [1] to evaluate the complexity. Let n be the number of keys currently stored in the table, and B be the maximum number of keys that fit inside one block transfer, where a `stash` of k keys can be kept in main memory. We measure space occupancy using the *space utilization* $1 - \epsilon$, where $0 \leq \epsilon < 1$, defined as the ratio of the number n of keys divided by the number of external-memory blocks times B, hence the number of blocks is $\lceil \frac{n}{B(1-\epsilon)} \rceil$. In other words, ϵ represents the "waste" of space in external memory, so the lower ϵ, the better.

Round-table achieves the following bounds. Each lookup reads just 1 block from external memory in the worst case, taking $O(1)$ CPU time and thus requiring only $O(1)$ words from main memory. Each update (insertion or deletion) requires to access at most $4s_0$ blocks in external memory, in the worst case, taking $O(s_0(B + \log n/\log\log n))$ CPU time w.h.p. (expected time is $O(s_0 B)$) and using $O(B)$ memory cells. The number of keys in the `stash` is $k \approx n/\exp(B)$. Experiments in Section 4 confirm our estimation.

In the literature for external-memory hashing, Mirrokni et al. [19] provide a version that keeps bucket load within a factor of $1 + \epsilon$, but cannot guarantee at most one memory access. The optimal bounds in Jensen and Pagh [12] and Conway et al. [6] do not require the stash, with no guarantee of at most one memory access. As for the work on tables with one external-memory access, some results [17, 9] rely on perfect hashing, but are either not dynamic or cannot reach arbitrarily high utilization. A recent cuckoo hashing based approach [21], combined with in-memory Bloom filters to ensure that lookups access the correct position, is not simple to dynamize. A general scheme [22] relies on perfect hashing to store the `stash` on external memory, thus having higher worst-case cost for insertions. The result in [5] achieves single-access lookups, but at the cost of $O\left(\frac{n}{B(1-\epsilon)}\right)$ internal memory. A solution based on predecessor search needs $O\left(\frac{n}{B}\right)$ internal memory, as discussed in [20].



(a) round $q = 0$ **(b)** round $q = 1$ **(c)** round $q = 2$ **(d)** round $q = 3$

Figure 1 Example of round-mapping with $s_0 = 3$, where the sequences of bucket numbers are not actually materialized by our algorithm. Black colored bucket numbers represent those buckets that have been added during round q.

2 Round-Mapping and Round-Hashing

Conceptually we map the range of our hash function onto a circle of unitary circumference, starting from a fixed point 0. For a given integer $s_0 > 1$, the circumference is then split into arcs of length proportional to either $1/s$ or $1/(s + 1)$ for some integer s ($s_0 \leq s \leq 2s_0 - 1$). We refer to arcs of length proportional to $1/s$ (resp. $1/(s + 1)$) as *long* (resp. *short*) arcs. At any time *all the short arcs, if any, appear consecutively* along the circumference, starting from point 0 and proceeding clockwise (hence, also the long arcs appear consecutively).

Each arc has a corresponding *arc number*, which is simply its position along the circumference, and *bucket number*, which is assigned at the moment of the creation of the arc and maintained implicitly as the algorithm progresses. All elements whose hash value fall inside a given arc are assigned to the corresponding bucket.

At the beginning, we set $s = s_0$ and start with s_0 long arcs. We also keep a counter for the number of buckets, initially zero.

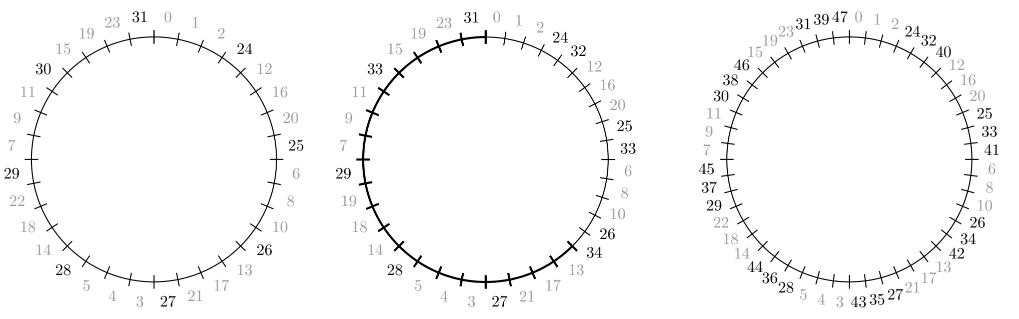
Operation `numBuckets()` simply returns the value of the above counter in $O(1)$ time.

Operation `findBucket`(u) is more involved, and is discussed in Section 2.1.

Operation `newBucket()` is implemented in $O(s_0)$ time by looking at long arcs, as follows. First, we check the border condition "all arcs are short": in that case, if $s < 2s_0 - 1$ then we set $s := s + 1$; else, we set $s := s_0$; either way, all the arcs become long. Second, we run *step* s to allocate a new bucket by taking the first s long arcs that are encountered clockwise along the circumference, and by replacing them with new $s + 1$ short arcs, say, $a_0, a_1, \ldots, a_s$. Their associated bucket numbers are mapped in the following way: the bucket numbers for $a_0, a_1, \ldots, a_{s-1}$ are inherited from the s long arcs that created them; the bucket number of a_s is the value of the counter, which is then increased by 1.

▶ **Lemma 1.** *At any time, the number of long arcs is always a multiple of s.*

Figures 1 and 2 show an example when a sequence of calls to `newBucket()` is performed (only some snapshots of the computation are presented). To understand the example, it helps to introduce a couple of concepts. We say that a *round* starts when the condition $s = s_0$ holds. Let the rounds be numbered as $q = 0, 1, 2, \ldots$, and let the length $len(q) = s_0\, 2^q$ of a round q represent the number of buckets at the end of its step $s = 2s_0 - 1$. For example, choosing $s_0 = 3$, the first rounds $q = 0, 1, 2, 3$ are shown in Figure 1. At step $s = 4$ of round $q = 4$, shown in Figure 2, each call to `newBucket()` takes s consecutive bucket numbers and inserts a new bucket number: after 0 1 2 24 it inserts 32, after 12 16 20 25 it inserts 33, after 6 8 10 26 it inserts 34, and so on. Note that 32, 33, 34, etc., are *native* of round $q = 4$ as



(a) end of step $s = s_0 = 3$ **(b)** between step $s = 3$ and 4 **(c)** end of step $s = 2s_0 - 1 = 5$

Figure 2 More detailed examples with $s_0 = 3$ during round $q = 4$. Long arcs are thicker.

Algorithm 1: Mapping from arcs to buckets.

```
Function findBucket(u)
    $j \leftarrow$ arc hit by $u$
    if $j < s_0$ then return $j$
    if $j > p$ then $j' \leftarrow j - \frac{p+1}{s+1}$, $s' = s$
    else $j' \leftarrow j$, $s' = s + 1$
    $x \leftarrow (j' \,\%\, s') \,\%\, s_0$
    $q' \leftarrow q + \left\lfloor \frac{s'-1}{s_0} \right\rfloor$
    $i = \left(1 + \left\lfloor \frac{s'-1}{s_0} \right\rfloor\right) \cdot \left\lfloor \frac{j'}{s'} \right\rfloor + \left\lfloor \frac{j' \% s'}{s_0} \right\rfloor$
    return $pos(i, x, q')$

Function $pos(i, x, q)$
    $e \leftarrow$ position of the least significant bit 1 in $i$
    return $\lfloor \frac{(s_0+x)2^q+i}{2^{e+1}} \rfloor$
```

they are created there. Black numbers in Figures 1 and 2 indicate which bucket numbers are native for the round. After step $s = 2s_0 - 1$, each round contains twice the bucket numbers than the previous round. Also, the concatenation of every other chunk of s_0 non-native bucket numbers, produces exactly the outcome of the previous round. We will exploit this regular pattern in the rest of the section.

Operation `freeBucket()` is simply the unrolling of the last `newBucket()` operation performed, hence, it also requires $O(s_0)$ time.

2.1 Implementation of `findBucket`(u)

We exploit the invariant property that short arcs are numbered from 0 to p, and thus $p+1$ is a multiple of $s+1$, where p is maintained as the last added short arc. We also use $pow(a)$, where $a > 0$, to denote the largest integer exponent $e \geq 0$ such that 2^e divides a (a.k.a. 2-adic order). Equivalently, $pow(a)$ is the position of the least significant bit 1 in the binary representation of the unsigned integer $a > 0$.

First, consider the ideal situation: after the step $s = 2s_0 - 1$ of round q, we have $len(q)$ buckets, numbered consecutively from 0 to $len(q) - 1$. We also have $len(q)$ arcs on the circle, numbered consecutively from 0 to $len(q) - 1$. As arc j is mapped to bucket number $b(j)$ using our scheme, we give a closed formula for $b(j)$ that can be computed in $O(1)$ time in the word RAM model, where divisions and modulo operations involve just powers of two or constants in the range $[s_0 \ldots 2s_0]$ (see Algorithm 1).

Let $j = s_0\, i + x$ where $x \in \{0, 1, \ldots, s_0 - 1\}$. If $i = 0$, then $b(j) = b(x) = x$. Thus $b(j) = j$ for $0 \leq j < s_0$. Hence, let assume $i > 0$ in the rest of the section, and thus we need to compute $b(j)$ for $j \geq s_0$.

We say that the bucket number in position j belongs to *chunk* i (hence, a chunk is of length s_0). For odd values of i, the bucket number is native for round q. For even values of i, the bucket number is native for round $q - pow(i)$, as it can be checked in Figure 1: for example, in round q after the last step, bucket number 9 is in position $j = 37 = 3 \cdot 12 + 1$, so $i = 12$ and 9 is native for round $q - pow(i) = 4 - 2 = 2$. In general, as $pow(i) = 0$ when i is odd, we can always say that the bucket number is native for round $q - pow(i)$ for $i > 0$. Another useful observation is that the smallest native number in round q is $len(q - 1)$ by construction (e.g. 24 in round $q = 4$).

In the ideal situation, we find the native round for the bucket number at position j: as its chunk is preserved in the native round, we can use its offset x inside the chunk to recover the value of that bucket number. In the native round q, each chunk i starts with bucket number $len(q - 1)$ as previously observed, increased by one for each such chunk, thus the first bucket number in chunk i is $len(q - 1) + \lfloor i/2 \rfloor$. Also, any two adjacent numbers in the chunk, differ by 2^{q-1} by construction. Summing up, there are two cases for the bucket number for j:

- i odd and thus native for round q: the bucket number is $\left\lfloor \frac{(s_0+x)2^q+i}{2} \right\rfloor$
- i even and thus native for round $q - pow(i)$: the bucket number is $\left\lfloor \frac{(s_0+x)2^q+i}{2^{pow(i)+1}} \right\rfloor$

As $pow(a) = 0$ when a is odd, we can compactly write these positions in the ideal situation as

$$pos(i, x, q) = \left\lfloor \frac{(s_0 + x)2^q + i}{2^{pow(i)+1}} \right\rfloor$$

Second, consider the general situation, with an intermediate step $s_0 \leq s \leq 2s_0 - 1$ in round q. Recall that we know the position p of the last created arc. This gives the following picture. The first $p + 1$ short arcs in clockwise order can be seen as $\frac{p+1}{s+1}$ consecutive groups, each of $s + 1$ arcs, and the remaining arcs are long and form groups of s arcs each. Let us set $s' = s + 1$ in the former groups, and $s' = s$ in the latter groups. In the following, we equally say that each group contains s' arcs or that each group contains s' bucket numbers. In general, we say s' entries (arcs or bucket numbers) when it is clear from the context.

A common feature is that the first s_0 entries of each group are inherited from the previous round, and the last $s' - s_0$ entries in each group are those added in the current round: each new entry is *appended* at the end of each group, so the entry in position p is the last in its group.

Now, given a position j, we want to compute $b(j)$, the corresponding bucket number. The idea is to reduce this computation to the ideal situation analyzed before.

If $j > p$, we conceptually remove one entry for each group such that $s' = s + 1$. This is equivalent to set $j := j - \frac{p+1}{s+1}$ and, consequently, $p := p - \frac{p+1}{s+1}$. Now, we have all the groups of the same size s', which are sequentially numbered starting form 0.

Let $i' = \lfloor j/s' \rfloor$ be the number of the group that contains the entry corresponding to j. We now decide whether j is one of the first s_0 entries of its group or not. We have two cases, according to the value of $r = j \,\%\, s'$.

If $r < s_0$, the wanted entry is one of the first s_0 entries of its group. If we concatenate those entries over all groups, we obtain the ideal situation of the previous round $q - 1$. There, the wanted entry occupies position $j' = s_0 i' + r$. Hence, $b(j) = pos(i', r, q - 1)$ in the ideal situation.

If $r \geq s_0$, the wanted entry is one of the last s' entries of its group. Analogously, if we concatenate those entries over all groups, the position of the wanted entry becomes

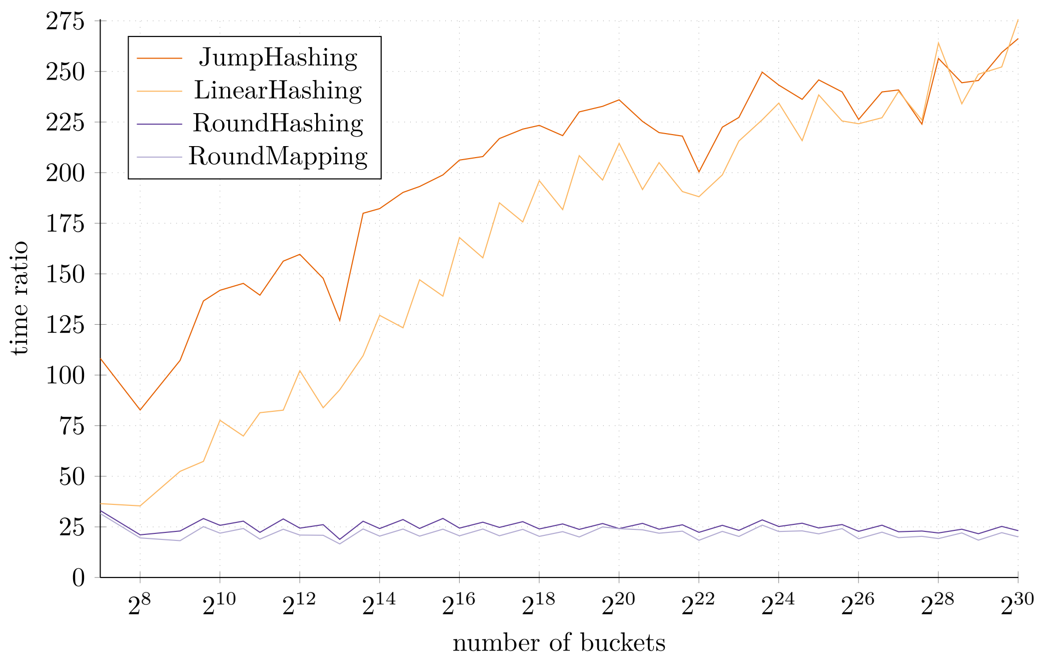


Figure 3 Time needed to compute a single hash as the number of buckets varies.

$j'' = (s' - s_0)i' + r - s_0$, where $x = r - s_0$ is the internal offset. However, we cannot solve this directly. We use instead the observation that the futures entries that will contribute to get the ideal situation for round q, will be *appended* at the end of each group. In this ideal situation, the wanted entry correspond to arc $2i'+1$ and is at position $j' = s_0(2i'+1)+r-s_0$ for round q. Thus, $b(j) = pos(2i'+1, r-s_0, q)$ in the ideal situation.

We can summarize the entire computation of $b(j)$ in an equivalent formula computed by Algorithm 1 that can be computed in $O(1)$ time.

▶ **Lemma 2.** `findBucket()` *can be implemented in* $O(1)$ *time using bitwise operations.*

Interestingly, `findBucket()` is much faster than other approaches known in the literature for consistent hashing, as we will see in the experiments.

▶ **Theorem 3.** *Round-mapping with integer parameter* $s_0 > 1$ *can be implemented using* $O(1)$ *words, so that* `init()`, `numBuckets()` *and* `findBucket()` *take* $O(1)$ *time, and* `newBucket()` *and* `freeBucket()` *take* $O(s_0)$ *time.*

Round-hashing computes the hash value of the given key and invokes round-mapping to obtain its bucket number. Whenever it decides to increase or decrease the number m of buckets, it invokes again round-mapping to know the $\Theta(s_0)$ buckets whose keys must be redistributed. Letting α be the load factor, namely, the overall number of stored keys divided m, we obtain the result in the last row of Table 1.

▶ **Theorem 4.** *Round-hashing with integer parameter* $s_0 > 1$ *requires* $O(1)$ *working space and takes* $O(1)$ *time to find the bucket for the key to be searched, inserted or deleted, and* $\tilde{O}(s_0\alpha)$ *average time to add or remove a bucket whenever needed.*

3 Distributed Servers

We experimentally evaluated round-hashing and our C implementation of Algortihm 1, on a commodity hardware based on Intel Xeon E3-1545M v5 CPU and 32Gb RAM, running Linux 4.14.34, and using `gcc` 7.3.1 compiler. We give some implementation details on the experimented algorithms, observing that we decided not to run consistent hashing [13] and rendezvous hashing [24] as they are outperformed by jump consistent hashing as discussed in detail in [14]. Specifically, we ran the following code.

- Jump consistent hashing [14]: we employed the implementation provided by the authors' optimized code.
- Linear hashing [15, 16]: the pseudocode is provided but not the code, which we wrote in C. As for the $O(\log m)$ hash functions, we followed the approach suggested in [15]: we employed the fast and high-quality pseudo-random number generator in [25] using the key to hash as a seed and the jth output as the outcome of the jth hash function. This takes constant time per hash function. Moreover, we replaced all modulo operations with the equivalent faster operations, as we did for round-mapping.
- Round-hashing (this paper): we employed the first output from the pseudo-random number generator in [25] as hash value. We chose the size of our hash range to be a power of two, so that mapping a hash value to an arc number can be done without divisions: we computed the product between the number of buckets and the hash value, divided by the maximum possible hash value. Note that some care is required to compute the product correctly as it may overflow.

It is worth noting that replacing the expensive division was very effective in our measurements. In particular, we replaced the division by s' in Algorithm 1 with the precomputed equivalent combination of multiplication and shift: as $s_0 \leq s' \leq 2s_0$, this can be done at initialization time with a constant amount of work. This reduced the time per round-hashing call from 14.02ns to 8.71ns, a 60% decrease, which is an interesting lesson that we learned.

Figure 3 shows the running times for the above implementations, when computing ten million hash values, as the number of buckets varies on the x-axis. On the y-axis, the running times are reported for jump consistent hashing, linear hashing, round-hashing and round-mapping alone (i.e. given a position u in the circumference, return its bucket number). As it can be seen, the costs of round-hashing and round-mapping are very close and *constant* along the x-axis, outperforming the non-constant costs of jump consistent hashing and linear hashing, which behave similarly when the number of buckets is large. Note that round-hashing has at least an order of magnitude improvement at around 2^{16} buckets and on, which indicates that it scales well.

All the running times in Figure 3 were normalized by the time needed to compute the sum of all the values. Looking at the absolute figures, the running time for the sum is about 0.4ns per element, and that of round mapping is 8–10ns per element (and the pseudo-random number generator in [25] takes twice the cost of the sum).

Speed is not the whole story as it is important also how the hash values in the range are distributed in the buckets. To this end, we show in Table 2 the results using 64-bit hash values: as it was infeasible to compute the bucket for every possible hash value, we chose 10^9 values at regular intervals in the hash range of 2^{64} values, and computed the bucket size distribution for them.

The columns in the table report the parameters for 10^4 buckets, where the actual bucket sizes are obtained by multiplying parameters in {min,max,1%,99%} by the load factor $\alpha = 10^9/10^4$. Specifically, s_0 useful for linear hashing and round-hashing, the standard

Table 2 Statistics on how much hash space is assigned to a given bucket, with a total of 10000 buckets. Note that the actual bucket sizes are obtained by multiplying the numbers in columns min, max, 1%, 99& by the load factor α. Extremal values and percentiles are a ratio from the ideal value.

	s_0	$\frac{\sigma}{\mu}$	min	max	1%	99%	percentile ratio
jump consistent h.		0.316	0.988	1.012	0.993	1.007	1.014
round-hashing	1	29.325	0.610	1.221	0.610	1.221	2.001
	2	20.272	0.814	1.221	0.814	1.221	1.500
	4	7.192	0.977	1.221	0.977	1.221	1.250
	8	4.465	0.976	1.085	0.976	1.085	1.112
	16	2.560	0.976	1.028	0.976	1.028	1.053
	32	0.613	0.976	1.002	0.976	1.002	1.027
	64	0.421	0.989	1.002	0.989	1.002	1.013
	128	0.277	0.995	1.002	0.995	1.002	1.007
linear hashing	1	29.329	0.602	1.232	0.605	1.228	2.030
	2	20.274	0.803	1.234	0.808	1.228	1.520
	4	7.203	0.964	1.232	0.969	1.225	1.264
	8	4.476	0.965	1.095	0.970	1.090	1.124
	16	2.583	0.965	1.041	0.970	1.034	1.066
	32	0.685	0.968	1.014	0.973	1.009	1.037
	64	0.527	0.980	1.014	0.984	1.009	1.025
	128	0.417	0.985	1.014	0.990	1.009	1.019

error $\frac{\sigma}{\mu}$ where σ is the variance and μ is the average of the bucket sizes, the minimum and maximum bucket size, the 1% and 99% percentiles of the size, and the ratio between the latter two. This ratio is the most important parameter in the table as it shows how well-balanced are buckets. It can be easily seen that both round-hashing and linear-hashing can match almost perfectly, with round-hashing having a slightly better distribution. Based on this table, we can see that round-hashing and linear-hashing have distribution properties that are similar to jump consistent hashing, as long as we choose suitable values: $s_0 = 64$ for round-hashing and $s_0 = 128$ for linear hashing. Figure 3 has been plotted using these values of s_0.

4 External-Memory Tables

Given a universe U of keys, and a random hashing function $h : U \to I$, where $I = \{0, 1, \ldots, |I| - 1\}$, we build a hash table that keeps a `stash` of keys in main memory. Armed with the round-hashing, we obtain a hash table called *round-table* that uses $O(k+1)$ words in main memory, where k denotes the number of stash keys. We consider the `stash` to be a set of k keys, where notation `stash`$[b]$ indicates the set $\{x \in \texttt{stash} : \texttt{findBucket}(h(x)/|I|) = b\}$ (e.g. a hash table in main memory with maximum size $O(B + \log n/\log\log n)$ w.h.p. via a classical load balancing argument). To check if $x \in \texttt{stash}$, we check if $x \in \texttt{stash}[b]$ where $b = \texttt{findBucket}(x)$. Also, for a user given parameter ϵ, the guaranteed space utilization in external memory is $1 - \epsilon$.

The lookup algorithm is straightforward while the insertion algorithm is a bit more complex. After checking that the key is not in the table, it proceeds with the insertion. For this, we need to maintain the claimed space utilization of $(1 - \epsilon)$. That is, if $\lceil \frac{n}{B(1-\epsilon)} \rceil >$

Table 3 Percentage of elements on the stash as s_0 and ϵ change, with $B = 1024$.

s_0	ϵ											
	0		0.001		0.01		0.03		0.05		0.1	
	real	est.	real	est.	real	est.	real	est.	real	est.	real	est.
1	17.2%	18.4%	17.1%	18.3%	16.7%	17.4%	15.9%	15.7%	15.1%	14.5%	13%	12%
4	5.6%	6.8%	5.5%	6.7%	5.1%	5.9%	4.2%	4.4%	3.4%	3.4%	1.7%	1.7%
16	1.8%	2.8%	1.7%	2.7%	1.3%	1.9%	0.7%	0.7%	0.3%	0.1%	**0.01%**	**0.1%**
32	1.4%	2%	1.3%	1.9%	0.9%	1.2%	0.4%	0.3%	**0.1%**	**0.5%**	**0.003%**	**0.009%**
64	1.3%	1.6%	1.2%	1.5%	0.8%	0.9%	**0.3%**	**0.6%**	**0.1%**	**0.2%**	**0.003%**	**0.002%**
256	1.3%	1.3%	1.2%	1.3%	**0.8%**	**1%**	**0.3%**	**0.3%**	**0.08%**	**0.09%**	**0.003%**	**0.0005%**
ideal	-	1.2%	-	1.2%	-	0.8%	-	0.3%	-	0.07%	-	0.0003%

numBuckets(), we need one more block. We invoke newBucket(), and receive a list of $z < 2s_0$ block numbers. We have to distribute the keys stored in these z blocks over $z + 1$ blocks, where the extra block has number numBuckets() as it is the latest allocated block number by round-mapping. In the distribution, the keys from the stash are also involved, as described below in the function distribute. After that, findBucket() finds the external-memory block block() that should contain the key: if it is full, the key is added to the stash.

Function distribute$(b_0, b_1, \ldots, b_{z-1})$ takes these z block numbers from newBucket(), knowing that $b_z =$ numBuckets() is the new allocated block number, and thus allocates block(b_z). Then it loads block(b_{z-1}) and moves to block(b_z) all keys $x \in$ block(b_{z-1}) such that findBucket$(x) = b_z$. Also, for each $x \in$ stash$[b_{z-1}]$ such that findBucket$(x) = b_z$, it moves x to block(b_z), if there is room, or to stash$[b_z]$ otherwise. Next, we repeat this task for b_{z-2} and b_{z-1} while also taking care of moving keys from stash$[b_{z-1}]$ to block(b_{z-1}) if there is room, and so on. In this way, the cost of distribute is $2z + 1$ block transfers, using $O(B)$ space in main memory, taking $O(s_0(B + \log n/\log\log n))$ CPU time w.h.p., and $O(s_0 B)$ expected time.

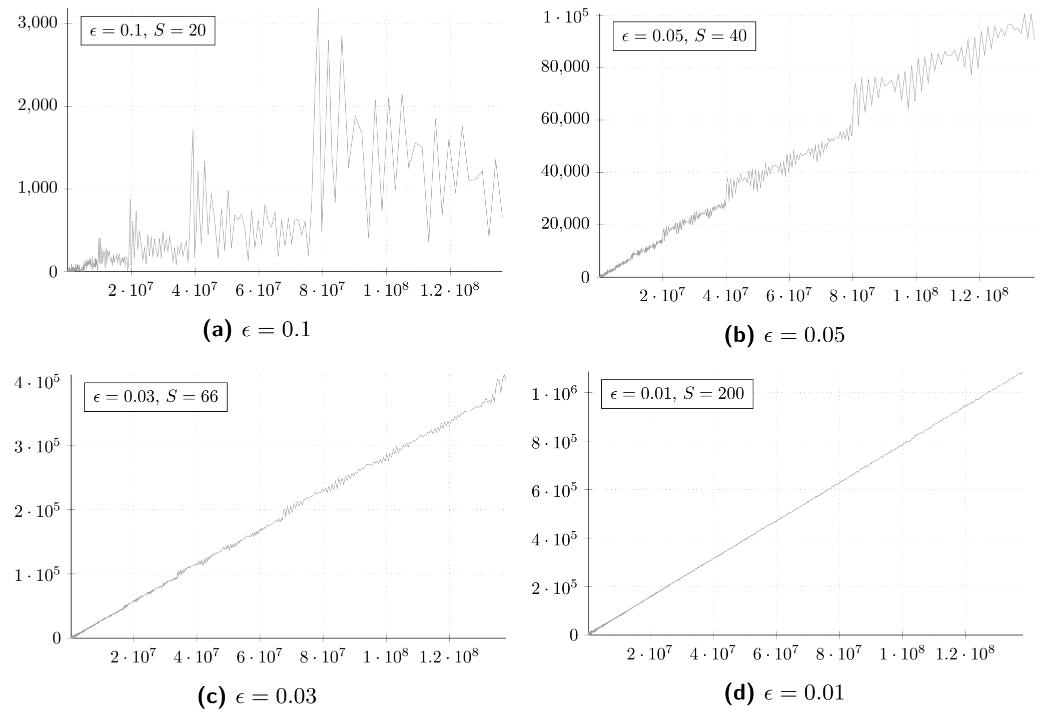
The deletion algorithm is similar to the insertion one, and its performance can be bound in the same way as above. We check the condition $\lceil \frac{n}{B(1-\epsilon)} \rceil <$ numBuckets() $- 1$ for $n > 0$ to run freeBucket() using a slightly different distribute that proceeds in reverse. Note that the rhs of the condition is numBuckets() $- 1$ to avoid newBucket() being called too soon.

We can show that as long as we choose $s_0 > \frac{2}{\epsilon}$ we have that the stash size of a hash table implemented with round-hashing is similar to the behaviour we would get with an uniform hash function (that would require rehashing). Thus, we recommend choosing $s_0\epsilon > 2$, as confirmed by the experiments below. Moreover, we can show how to keep a copy of the stash in external memory, without increasing space usage but increasing the number of block operations per update to $O(1 + \epsilon s_0)$.

To evaluate our approach, we consider the worst-case stash size (over the number of keys) across multiple values of n (going from $2^{10}B$ to $2^{13}B$) for $B = 512, 1024, 2048$ as ϵ and s_0 vary. The results are reported in Tables 3, where the left side of every column reports the ratio predicted by the analysis and the right side shows the effective maximum ratio reported during the experiment. As our analysis is substantially different when $\epsilon s_0 > 1$, we reported those values in bold to highlight them. Finally, the last row reports the best values one can hope to achieve for that value of ϵ, that is, the values that our analysis predicts for a uniform hash function.

Looking at these results, we can make some observations. First, the values predicted by the analysis match the results fairly well, especially when $s_0\epsilon \gg 1$ or $s_0\epsilon \ll 1$. In particular, it almost never happens that the analysis is wrong by more than a factor of 3. Second, when

Figure 4 Stash size (on the y-axis) as n grows (on the x-axis) for $s_0 = \frac{\epsilon}{2}$ and different values of ϵ.



s_0 is small, stash size is fairly high, even for low space utilization. This is to be expected, as in this case different buckets may have very different assignment probabilities. Third, as s_0 grows, stash size quickly approaches the one that we would expect from the ideal case. Nonetheless, the improvement is fairly small when s_0 goes over 32, even at low utilization. We thus recommend s_0 to be chosen near 32 for practical usage.

We also considered how stash size varies over time, as more elements are inserted. To study that, we fixed $s_0 = \frac{2}{\epsilon}$, as recommended in the analysis section, and plotted the size of the `stash` against the number of elements in the table. The plots can be found in Figure 4. These plots clearly show the "cyclic" behavior of round-table: when a new round begins, the distribution of keys in buckets is further away from being uniform and, as a result, the stash size increases. As more steps of the round are completed, the spikes in stash size get progressively smaller as round-table balances keys in a better way, until a new round starts again and the table reverts to its previous behavior.

5 Conclusions

We discussed a version of consistent hashing, called round-hashing, that scales well for large data sets in distributed servers. A key tool is round-mapping, and it would be interesting to see if it can have other applications, and if the number of changed buckets at each step can be reduced while keeping the same guarantees. As an example, we discussed how to obtain a dynamic hash table for external memory that guarantees at most one access to the external memory in the worst case, $(1 - \epsilon)$ space utilization, efficient updates, and small stash size in main memory.

References

1 Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, sep 1988.

2 A. Andersson, P. B. Miltersen, S. Riis, and M. Thorup. Static dictionaries on AC^0 RAMs: query time $\theta(\sqrt{\log n/\log\log n})$ is necessary and sufficient. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 441–450, Oct 1996. `doi:10.1109/SFCS.1996.548503`.

3 Noa Bar-Yosef and Avishai Wool. Remote algorithmic complexity attacks against randomized hash tables. In *SECRYPT 2007, Proceedings of the International Conference on Security and Cryptography, Barcelona, Spain, July 28-13, 2007, SECRYPT is part of ICETE - The International Joint Conference on e-Business and Telecommunications*, pages 117–124, 2007.

4 Andrej Brodnik, Peter Bro Miltersen, and J. Ian Munro. Trans-dichotomous algorithms without multiplication—some upper and lower bounds. In *Algorithms and Data Structures, 5th International Workshop, WADS '97, Halifax, Nova Scotia, Canada, August 6-8, 1997, Proceedings*, pages 426–439, 1997.

5 F. Cesarini and G. Soda. Single access hashing with overflow separators for dynamic files. *BIT Numerical Mathematics*, 33(1):15–28, Mar 1993. `doi:10.1007/BF01990340`.

6 Alexander Conway, Martin Farach-Colton, and Philip Shilane. Optimal hashing in external memory. In *Proc. Automata, Languages and Programming, 45th International Colloquium (ICALP18)*, 2018.

7 Scott A. Crosby and Dan S. Wallach. Denial of service via algorithmic complexity attacks. In *USENIX Security Symposium*. USENIX Association, 2003.

8 Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, pages 205–220, 2007.

9 Ronald Fagin, Jurg Nievergelt, Nicholas Pippenger, and H. Raymond Strong. Extendible hashing - a fast access method for dynamic files. *ACM Trans. Database Syst.*, 4(3):315–344, 1979. `doi:10.1145/320083.320092`.

10 T. Granlund and P. L. Montgomery. Division by invariant integers using multiplication. *ACM Sigplan Notices*, 29:61–72, 1994.

11 Intel Co. *Intel 64 and IA-32 Architectures Optimization Reference Manual*. Intel, order 248966-040, April 2018.

12 Morten Skaarup Jensen and Rasmus Pagh. Optimality in external memory hashing. *Algorithmica*, 52(3):403–411, 2008.

13 David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 654–663, New York, NY, USA, 1997. ACM. `doi:10.1145/258533.258660`.

14 John Lamping and Eric Veach. A fast, minimal memory, consistent hash algorithm. *CoRR*, abs/1406.2294, 2014. `arXiv:1406.2294`.

15 Per-Åke Larson. Linear hashing with partial expansions. In *VLDB*, volume 6, pages 224–232, 1980.

16 Witold Litwin. Linear hashing: a new tool for file and table addressing. In *VLDB*, volume 80, pages 1–3, 1980.

17 Harry G Mairson. The program complexity of searching a table. In *Foundations of Computer Science, 1983., 24th Annual Symposium on*, pages 40–47. IEEE, 1983.

18 Peter Bro Miltersen. Lower bounds for static dictionaries on RAMs with bit operations but no multiplication. In *Automata, Languages and Programming, 23rd International Colloquium, ICALP96, Paderborn, Germany, 8-12 July 1996, Proceedings*, pages 442–453, 1996.

19 Vahab Mirrokni, Mikkel Thorup, and Morteza Zadimoghaddam. Consistent hashing with bounded loads. *CoRR*, 2016. URL: `https://arxiv.org/pdf/1608.01350v1`.

20 Rasmus Pagh. Basic external memory data structures. *Algorithms for Memory Hierarchies*, pages 14–35, 2003.

21 Salvatore Pontarelli, Pedro Reviriego, and Michael Mitzenmacher. EMOMA: exact match in one memory access. *CoRR*, abs/1709.04711, 2017. `arXiv:1709.04711`.

22 M. V. Ramakrishna and Walid R. Tout. *Dynamic external hashing with guaranteed single access retrieval*, pages 187–201. Springer Berlin Heidelberg, Berlin, Heidelberg, 1989. `doi:10.1007/3-540-51295-0_127`.

23 Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

24 David Thaler and Chinya V. Ravishankar. Using name-based mappings to increase hit rates. *IEEE/ACM Trans. Netw*, 6(1):1–14, 1998.

25 Sebastiano Vigna. `xoroshiro128+`: an extremely fast and well-distributed pseudo random number generator. `http://xoroshiro.di.unimi.it/`, 2018.

26 Wei Wang and Chinya V. Ravishankar. Hash-based virtual hierarchies for scalable location service in mobile ad-hoc networks. *Mobile Networks and Applications*, 14(5):625–637, 2009.

Algorithmic Building Blocks for Asymmetric Memories

Yan Gu
Carnegie Mellon University, Pittsburgh, PA, USA
yan.gu@cs.cmu.edu

Yihan Sun
Carnegie Mellon University, Pittsburgh, PA, USA
yihans@cs.cmu.edu

Guy E. Blelloch
Carnegie Mellon University, Pittsburgh, PA, USA
guyb@cs.cmu.edu

Abstract

The future of main memory appears to lie in the direction of new non-volatile memory technologies that provide strong capacity-to-performance ratios, but have write operations that are much more expensive than reads in terms of energy, bandwidth, and latency. This asymmetry can have a significant effect on algorithm design, and in many cases it is possible to reduce writes at the cost of more reads. This paper studies which algorithmic techniques are useful in designing practical write-efficient algorithms. We focus on several fundamental algorithmic building blocks including unordered set/map implemented using hash tables, comparison sort, and graph traversal algorithms including breadth-first search and Dijkstra's algorithm. We introduce new algorithms and implementations that can reduce writes, and analyze the performance experimentally using a software simulator. Finally, we summarize interesting lessons and directions in designing write-efficient algorithms that can be valuable to share.

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

Keywords and phrases Asymmetric Memory, I/O Cost, Write-Efficient Algorithms, Hash Tables, Graph-Traversal Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.44

Related Version The full version is available at [20], `https://arxiv.org/abs/1806.10370`.

Acknowledgements This work was supported by NSF grants CCF-1408940, CCF-1533858, CCF-1629444 and CCF-1745331.

1 Introduction

The future of main memory appears to lie in the non-volatile memory technologies that promise persistence, significantly lower energy costs, and higher density than the DRAM technology used in today's main memories [21, 24, 33, 43]. However, despite the advantages, a key property of such memory technologies is their asymmetric read-write costs: compared to reads, writes can be much more expensive in terms of latency, bandwidth, and energy. Because bits are stored in these technologies as at rest "states" of the given material that can be quickly read but require physical change to update, this asymmetry appears fundamental.

© Yan Gu, Yihan Sun, and Guy E. Blelloch;
licensed under Creative Commons License CC-BY

This motivates the need for *write-efficient* algorithms that largely reduce the number of writes compared to existing algorithms.

In the related work section, we review the literature on studying this read-write asymmetry on NAND Flash chips [4, 17, 18, 35] and algorithms targeting database operators [12, 39, 40]. These works provide novel aspects on rethinking algorithm design. However, most of the papers either treat NVMs as external memories, or are based on hardware simulators for existing architecture, which may have many concerns that we will further discuss in the related work section.

Blelloch et al. [5, 7, 8] formally defined and analyzed several sequential and parallel computation models with good caching and scheduling guarantees. The models abstract such asymmetry between reads and writes, and can be used to analyze algorithms on future memory. The basic model, which is the Asymmetric RAM (ARAM), extends the well-known external-memory model [1] and parameterizes the asymmetry using ω, which corresponds to the cost of a write relative to a read to the non-volatile main memory. The cost of an algorithm on the ARAM, the **asymmetric I/O cost**, is the number of write transfers to the main memory multiplied by ω, plus the number of read transfers. This model captures different system consideration (latency, bandwidth, or energy) by simply plugging in different values of ω, and also allows algorithms to be analyzed theoretically. Based on this idea, many interesting algorithms (and lower bounds) are designed and analyzed by various recent papers [5, 6, 7, 8, 10, 25].

Unfortunately, all of the analyses of such write-efficient algorithms are asymptotic, showing the upper and lower bounds on the complexity of these problems. Also, to prove the bounds, the theoretical models simplify the real architecture (e.g., without considering blocking of cache-lines or cache policies). It still remains unknown what the performance of these algorithms are in practice. In this paper, our goal is to show such performance on a number of fundamental algorithmic building blocks. We believe the lessons in designing and implementing them are useful for our community to use new memory in the future.

Contribution of this paper

In this work, our goal is to bridge the gap between theory and practice. We try to study and understand which algorithmic techniques are useful in designing practical write-efficient algorithms. As the first paper of this kind, we focus on several of the most commonly-seen algorithmic building blocks in modern programming. Due to the page limit, in this paper we briefly discuss unordered set/map implemented using **hash tables**, and graph traversal algorithms: **breadth-first search** for unweighted graphs and **Dijkstra's algorithm** for weighted graphs. In the full version of this work, we discuss more details of these algorithms, as well as ordered set/map implemented using **binary search trees** and **comparison sort**.

Unfortunately, no non-volatile main memory is currently available, making it impossible to get real timings. Furthermore, details about latency and other parameters of the memory and how they will be incorporated into the architecture are also not available. This makes detailed cycle-level simulation (e.g., PTLsim [36], MARSSx86 [34] or ZSim [38]) of questionable utility. However, it is quite feasible to count the number of reads and write to main memory while simulating a variety of cache configurations. For I/O-bounded algorithms, these numbers can be used as reasonable proxies for both running time (especially when implemented in parallel) and energy consumption.[1] Moreover, conclusions drawn from these numbers can likely give insights into tradeoffs between reads and writes among different algorithms.

[1] The energy consumption of main memory is a key concern since it costs 25-50% energy on data centers and servers [28, 32, 30].

For these reasons, we propose a framework based on a software simulator that can efficiently and precisely measure the number of read and write transfers of an algorithm using different caching policies. We also consider variants in caching policies that might lead to improvements when read and write are not the same.

We also note that designing write-efficient algorithms falls in a high dimensional parameter space since the asymmetries on latency, bandwidth, and energy consumption between reads and writes are different. Here we abstract this as a single value ω. This value together with the cache size M and cache-line size B (set to be 64 bytes in this paper) form the parameter space of an algorithm.

Our framework provides a simple, clean and hardware-independent method to analyze and experiment the performance on the asymmetric memory. We investigate the algorithmic techniques and learn lessons from the experiments that generally apply for a reasonably large parameter space of ω, M and B. This framework also allows monitoring, reasoning and debugging the code easily, so it can remain useful even after the new hardware is available.

With the framework, we design, implement and discuss many algorithms and data structures and their write-efficient implementations. Although some of the implementations are standard, like quicksort and hash tables, many others, including k-level hash tables, sample sort and phased Dijkstra, require careful algorithmic design, analysis, and coding. Under our measurement which is the asymmetric I/O cost and compared to the most commonly-used ones on symmetric memories, we provide better alternatives to all problems we studied in this paper.

With the algorithms and their experimental results, we draw many interesting algorithmic strategies and guidance in designing write-efficient algorithms. A common theme is to trade (more) reads for (fewer) writes (apparently it is hard to directly decrease the writes since this can improve the performance on symmetric memory as well and should have been investigated already). Some interesting lessons we learned and can be valuable to share are listed as follows, which can suggest some potential directions to design and engineer write-efficient algorithms in the future.

1. Indirect addressing is less problematic. In the classic setting, indirect addressing should be avoided if possible, since each addressing can be a random access to the memory. However, when writes are expensive, moving the entire data is costly, while indirect addressing only modifies the pointers (at the cost of a possible random access per lookup).
2. Multiple candidate positions for a single entry in a data structure can help. It can be a good option to use more reads per lookup but apply less frequent data movements, when the size of a data structure changes significantly. This is a common strategy we have applied in this paper to provide an algorithmic tradeoff between reads and writes.
3. It is usually worth to investigate existing algorithms that move or modify the data less. These algorithms can be less efficient in the symmetric setting due to various reasons (e.g., more random accesses, less balanced), but the property that they use fewer writes can be useful in the asymmetric setting (like samplesort vs. quicksort, treap vs. AVL or red-black tree).
4. In-cache data structures should draw more attention. Since the data structures are kept in the cache (or small symmetric memory), the algorithm requires significantly less writes to the large asymmetric memory, although may require extra reads to compensate for less information we can keep within the data structure. In this paper, we discuss Dijkstra's algorithm on shortest-paths as an example, and such idea can also be applied to computing minimum spanning tree, sorting, and many other problems.

2 Related Work

There exist a rich literature to show the read-write asymmetry on the new memories [2, 3, 7, 8, 11, 13, 15, 16, 22, 23, 26, 27, 31, 37, 41, 42, 44, 45]. Regarding adapting softwares for such read-write asymmetry, some work has studied the system aspect. For example, there exist many papers on how to balance the writes across the chip to avoid uneven wear-out of locations in the context of NAND Flash chips [4, 17, 18, 35].

The early and inspirational attempts to design algorithms with fewer writes targeting database operators: Chen et al. [12] and Viglas [39, 40] presented several write-efficient sequential algorithms for searching, hash joins and sorting. However, their results are mainly shown by assuming external memories rather than main memories, or on the cycle-based simulators for existing architecture. For the latter case however, the prototypes of the new memories are still under development, and yet nobody actually knows the exact parameters of the new memories, or how they are incorporated into the actual architecture. As a result, we believe that the results based on cycle-based simulator might not be very accurate. In the meantime, the asymmetries on latency, bandwidth, and energy consumption between reads and writes are different, and any of these constraints can be the bottleneck of an algorithm. Hence, designing algorithms on asymmetric memory are in a multiple-dimension parameter space, rather than just recording the running time from a simulator. Therefore, it is essential to develop theoretical models and tools that account for, and abstract this asymmetry and use them to analyze algorithms on future memory.

Blelloch et al. [5, 7, 8] formally defined several sequential and parallel computation models that take asymmetric read-write costs into account. Based on the computational models, many interesting algorithms (and lower bounds) are designed and analyzed in both sequential and parallel settings, which includes sorting, permuting, matrix multiplication, FFT, list/tree contraction, BFS/DFS and other graph algorithms, and many computational geometric and dynamic programming problems [5, 6, 7, 8, 10, 25, 9, 19]. Carson et al. [11] also presented write-efficient sequential algorithms for a similar model, as well as write-efficient parallel algorithms (and lower bounds) on a distributed memory model with asymmetric read-write costs, focusing on linear algebra problems and direct N-body methods. Although many problems under the asymmetric setting have been studied, all the analyses are asymptotic and only show the upper and lower bounds on the complexity of these problems.

3 Our Model and Simulator

To start with, we discuss how to measure the performance of algorithms on asymmetric memories. We begin with the computational model that estimates the cost of an algorithm. This model requires the numbers of read and write transfers between the non-volatile memory and the cache, so later we introduce how the numbers of an algorithm can be simulated.

The Cost Model for Asymmetric Memory. The most commonly-used cost measure of an algorithm is the time complexity based on the RAM model, which is the overall number of instructions and memory accesses executed in this algorithm. Nowadays, since the latency of an memory access is at least two orders of magnitudes more expensive than a CPU instruction, the *I/O cost* based on the external-memory model [1] is widely used to analyze the cost of an I/O-bounded algorithm. This model assumes a *small-memory* (cache) of size $M \geq 1$, and a unbounded-size *large-memory*. Both memories are organized in blocks (cache-lines) of B words. The CPU can only access the small-memory (with no cost), and it takes unit cost to transfer one block between the small-memory and the large-memory. This cost measure estimates the running time reasonably well for I/O-bounded algorithms, especially in multi-core parallelism. An efficient algorithm in practice should achieve optimality in both time complexity and I/O cost.

To account for more expensive writes on future memories, here we adopt the idea of an (M, ω)-Asymmetric RAM (ARAM) [8]: similar to the external-memory model, transferring a block from large-memory to small-memory takes unit cost; on the other direction, the cost is either 0 if this block is clean and never modified, or $\omega \gg 1$ otherwise. The **asymmetric I/O cost Q** of an algorithm is the overall costs for all memory transfers. We abbreviate such cost Q as the *I/O cost* throughout the paper, unless stated otherwise explicitly. Theoretical results on this new model have been studied in [5, 6, 7, 8, 10, 25, 9, 19].

Cache Policies. Either the classic external-memory model or the new ARAM assumes that we can explicitly manipulate the cache in the algorithm. This largely simplifies the analysis, and in many cases is provably within a constant factor of a more realistic cache's performance. For example, the standard least-recent used (LRU) policy is 2-competitive against the optimal offline cache-replacement sequence. However, the competitive ratio does not hold in the asymmetric setting in the worst case. The overhead is proportional to ω, which can be significant and problematic. In the full version of this paper [20], we discuss several alternative solutions with worst-case performance guarantees. In this conference version we show our experiment results based on the LRU policy, and the comparison to other policies are covered in the full version of this paper.

The Cache Simulator. To capture the number of reads and writes to the main memory, we developed a software simulator that can adapt to different cache policies. The cache simulator is composed of an ordered map that keeps tracks of the time stamp of the last visit to each cache-line in the current cache, and an unordered map that stores the mapping from each cache-line to the corresponding location in the ordered map if this cache-line is currently in the cache. Interestingly, the implementation of this cache simulator is a natural application of the techniques discussed in this paper.

The cache simulator encapsulates a new structure ARRAY that is used in coding algorithms in this paper. It is like a regular array that can be dynamically allocated and freed, and supports two functions: READ and WRITE to a specific location in this array. The ARRAYs are responsible for reporting the memory accesses of the algorithm to the cache simulator, and the cache simulator will update the state of the cache accordingly. Therefore, coding using the ARRAYs is not different from regular programming much.

The memory accesses to loop variables and temporary variables are ignored, as well as the call stack. This is because the number of such variables is small in all of the algorithms in this paper (usually no more than 10). Meanwhile, the call stack of all algorithms in this paper has size $O(\log n)$. The overall amount of uncaptured space is orders of magnitudes smaller than the amount of fast memory in our experiments.

The cache simulator maintains two counters: the number of **read transfers**, and the number of **write transfers**. When testing each algorithm on a specific input instance, the cache is emptied at the beginning and flushed at the end. A read or write is free if the location is already in the cache; otherwise, the corresponding cache-line is loaded, the counter of read transfer increments by 1, and the least-recently-used cache-line in this pool is evicted. Also, a write will mark the dirty-bit of the cache-line to be `true`. When evicting a dirty cache-line, the counter of write transfer increments by 1. Notice that memory reads can cause write transfers, and memory writes can lead to read transfers.

When simulating the **Classic** policy (i.e., the standard one), we also verified our simulated results to ZSim (cycle-level simulator for current architecture), and the numbers always differ by no more than 10% when the parameters are set correctly.

4 Unordered Sets and Maps

Sets and maps are two of the most commonly-used data types in modern programming. Most programming languages either have them built in as basic types (e.g., python) or supply them as standard libraries (C++, C#, Java, Scala, Haskell, ML). In this section, we discuss efficient implementations of unordered sets and maps implemented using hash tables.

Our implementation of unordered sets and maps is based on hash tables that support **lookup**, **insertion**, and **deletion**. The hash tables discussed in this section use open addressing and linear probing, since the goal of the data structure is to try to minimize the I/O cost focusing on smaller entries (accessing and reading larger entries are costly anyway so different hash-table implementations make minor differences). For simplicity, we assume no duplicate keys, and it is straightforward to handle the duplicates with minor modifications. In this setting, each operation of the hash table reads a small number of cache-lines, and an insertion or deletion will modify exactly one cache-line that contains the location of the key and will be eventually written back to the large-memory.

The challenge emerges when the set size changes dynamically. For an efficient implementation, we hope the overall size of the hash table to be neither too large nor too small. If the load factor passes 80%, linear probing's performance drastically degrades. On the other hand, we want the hash table size to be reasonably small to better utilize the small-memory (cache), since each cache-line holds more entries in this case. In practice, some implementations keep the load factor up- and lower-bounded by some constant. For example, a typical implementation keeps the occupancy of the hash table between 1/8 and 1/2, and the size doubles or shrinks by half if the number of entries exceeds this range. Such resizing reinserts p entries after at least $p/2$ insertions and deletions (where p is the set/map size). When reads and writes have approximately the same cost, the extra cost for such resizing is small compared to the query and update costs (e.g., the queries read from lots of memory locations). In the asymmetric setting however, the reads cost much less, but the extra writes in resizing can be significant: the resizing can incur at most twice ($p/(p/2) = 2$) the writes compared to the initial insertions (3× writes in total). Hence, our goal is to discuss an alternative approach that optimizes such extra writes.

4.1 The k-level Hash Table

Instead of keeping one hash table, our main idea is to maintain a small number k of hash tables simultaneously, where k is a pre-determined parameter. In particular, the k-level hash table *HashTable* is initialized with k arrays $\mathit{HashTable}_{1,\cdots,k}$ with size $2^{c'+i}$ for $1 \le i \le k$ (or smaller in specific applications) and a constant c'. In practice we set c' to be 5.

For insertions, when the overall load factor exceeds some threshold r, we allocate a new chunk of memory with the double size of the largest current array, and the smallest hash table is discarded after all elements in it have been reinserted back. Similarly for deletions, if the occupancy of the hash tables drops below a threshold l, a small array with half size of the current smallest hash table is allocated, and the largest table is freed after the entries in it being reinserted. For instance, a valid k-level hash table may contain two arrays of size $2^{15} = 32768$ and $2^{16} = 65536$, when $k = 2$ and 30000 entries in the current configuration. We show the pseudocode of the k-level hash table in Algorithm 1. The occupancy range

Algorithm 1: The k-level hash table.

```
Input: Parameter $k$, occupancy range $l$ and $r$
function LOOKUP($x$)
    for $i \leftarrow 1$ to $k$ do
        $p \leftarrow HashTable_i$.LOOKUP($x$)
        if $p \neq$ null then return $(i, p)$
    return null

function INSERT($x$) // $x$ is not in $HashTable$
    for $i \leftarrow 1$ to $k$ do
        if $HashTable_i.occupancy < r$ then
            $HashTable_i$.INSERT($x$)
            return
    Allocate $HashTable_{k+1}$ of size $2 \cdot HashTable_k.size$
    Relabel the hash tables with indices from 0 to $k$
    foreach $y \in$ HashTable$_0$ do
        INSERT($y$)
    Free $HashTable_0$

function DELETE($x$; $i$, $p$) // $x$ is located $p$-th in $HashTable_i$
    $HashTable_i$.DELETE($x, p$)
    if Overall occupancy is less than $l$ (and $HashTable_1.size > 1$) then
        Allocate $HashTable_0$ of size $HashTable_1.size/2$
        Relabel the hash tables with indices between 1 to $k+1$
        foreach $y \in$ HashTable$_{k+1}$ do
            INSERT($y$)
        Free $HashTable_{k+1}$
```

$0 < l < r < 1$ indicates when the resizing happens (an example of l and r can be $1/8$ and $1/2$). A classic implementation can be viewed as the special case of the k-level hash table when $k = 1$.

We now analyze the I/O cost Q of the k-level hash table. Here we assume that the size of the k-level hash table is larger than the small-memory and $1 - r < 1/B$, so on average, one lookup, insertion or deletion in a single level in the hash table requires no more than $c < 2$ cache-line loads to locate the position.

Lookup. In a k-level hash table, a lookup requires ck instead of c read transfers (c is the constant just defined) in the worst case (can quit earlier once the entry is found). The cost increases by a factor of k at most.

Insert. There are two definitions of insertions: an insertion that the key is known to be not in the set/map, or an insertion that it is unknown whether the key is in this set/map. Both cases are commonly-used. In this paper, we take the first definition and analyze the cost of this type of insertions. The second type of insertion can be viewed as a lookup first, then an insert if the lookup fails.

When inserting an element in a k-level hash table, we always try the smaller tables first. Once all tables are full, we resize it. More details can be found in Algorithm 1.

The I/O cost Q of an insertion comes in two parts: the cost of the initial insertion to the hash table, and the cost of this entry in future hash-table resizings. The cost of the initial insertion is no more than $c + \omega$, where c is the number of cache-line reads to find the position to insert, plus ω, one cache-line write for the actual insertion. The cost of resizing is more complicated to analyze.

We note that although a specific entry can be reinserted multiple times during different resizing processes, the overall number of element reinsertion is bounded, and thus we can a amortize the work. A resizing occurs when an insertion comes in and the hash table contains exactly $r \cdot 2^p(2^k - 1)$ elements for some positive integer p. In this case, at most $r \cdot 2^p$ entries (the size of the smallest hash table), are reinserted during the resizing. The total number of insertions from the last resizing is at least $r \cdot 2^{p-1}(2^k - 1)$ (assuming $4l \le r$), so the amortized I/O cost Q of reinsertion for each insertion is upper bounded by $\dfrac{(c+\omega)r \cdot 2^p}{r \cdot 2^{p-1}(2^k-1)} = (c+\omega) \cdot 2/(2^k - 1)$.

In the asymmetric setting when $\omega \gg 1$, the I/O cost of each insertion is approximately $\omega \cdot (1 + 2/(2^k - 1))$, indicating that compared to the classic implementation where $k = 1$, in the worst-case the improvement when $k = 2, 3, 4$ is about 44%, 57% and 62% respectively. The asymptotic improvement when $k \to +\infty$ is 67% ($\frac{2}{3}$).

Delete. A deletion in the k-level hash table is similar to an insertion except that a lookup for the location is required (details in Algorithm 1). The cost of the initial deletion is $ck + \omega$. A resizing of the hash table can occur after at least $l \cdot 2^p(2^k - 1)$ deletions for some positive integer p, and the current hash table keeps $l \cdot 2^p(2^k - 1)$ entries. However, it is possible that all of these entries are in the last hash table so they are all reinserted. We note that when reinserting the elements from the discarded array, we always try smaller arrays first. This means that a reinserted entry, if not being deleted in the future, will not be reinserted again in the next $\min(k - 1, \log_2 r/2l)$ shrinking resizings. Namely, the amortized extra cost of a deletion in future resizings is about ω/k if l is set to be about $2^{-k}r$. The overall I/O cost for a deletion is $Q = ck + \omega(1 + 1/k)$.

We have bounded of the I/O cost of each lookup, insertion or deletion, and the overall cost Q can be estimated by summing the amount of each operation multiplied by the cost of this operation. In practice, insertions and deletions can interleave. For example, when a deletion comes after an insertion, the number of entries remains the same, which leads to no further cost for these two updates afterward. The exact cost is also affected by the pattern of the sequence of the operations, and we will show by experiments.

4.2 Experiments

We provide the full experiment of our k-level hash table in the full version of this paper [20]. We test the performance on various update/query patterns, and report the numbers of read transfers and write transfers, as well as I/O costs. We also justify our result by comparing to the wall-clock running time (in the full paper [20]). Due to the space limit, in this conference version we only show one of the experiments here that contains insertions and queries.

In all experiments, we insert 1 million elements to an empty hash table. Each of the element is a 4-byte integer, and we vary the number of queries. The simulated cache contains 10,000 cache-lines. The occupancy rate is set to be $l = 0.2$ and $r = 0.8$. We have tried other parameters (r between 0.6 and 0.8 and $l = r/4$). The results slightly vary, but all general conclusions in this section still hold.

Many applications, like webpage caching or the breadth-first searches, only insert but never delete elements in a hash table. Our experiment starts with this simpler case. We first show the relationship between k (the number of hash tables) and the numbers of read transfers and write transfers for a variety of insertion/query ratios, and the results are shown in Table 1. We fix the number of insertions to be one million, and query α times after each insertion. We vary α from 0, 1/8, to 8 ($\alpha < 1$ indicates one query per $1/\alpha$ insertions). About

Table 1 Numbers of read and write transfers of the k-level hash tables with different query/insert ratios. Numbers of read and write transfers are divided by 10^6 (i.e., per insertion).

10^6 insertions, $\alpha \times 10^6$ queries where α is from 0 to 8, the cache contains 10,000 cache-lines.

α	0		1/8		1/4		1/2		1		2		4		8	
	RT	WT	RT	WT	RT	WT	RT	WT	RT	WT	RT	WT	RT	WT	RT	WT
k=1	1.35	1.17	1.44	1.18	1.52	1.19	1.69	1.21	2.02	1.24	2.68	1.27	4.00	1.31	6.64	1.34
k=2	0.85	0.79	1.06	0.84	1.23	0.87	1.54	0.91	2.09	0.96	3.11	1.00	5.07	1.03	8.94	1.05
k=3	0.76	0.72	1.08	0.80	1.32	0.85	1.73	0.90	2.44	0.95	3.76	0.99	6.31	1.02	11.32	1.05
k=4	0.70	0.67	1.11	0.78	1.40	0.82	1.89	0.88	2.74	0.93	4.30	0.97	7.33	1.00	13.31	1.03

Table 2 The I/O costs of the k-level hash tables with different query/insert ratios. The write-read ratio ω are selected to be typical projected values 10 (latency, bandwidth) and 100 (energy). Results are based on the numbers in Table 1. The numbers in red with underlines indicate the best choice of k that minimizes the I/O cost in this setting, and numbers in blue indicate better I/O costs compared to the classic hash table implementation (i.e., $k = 1$).

The I/O costs of the k-level hash tables with the same configurations in Table 1.

	$\omega = 10$								$\omega = 100$							
α	0	1/8	1/4	1/2	1	2	4	8	0	1/8	1/4	1/2	1	2	4	8
k=1	13.0	13.2	13.4	13.8	14.4	15.4	17.1	20.0	117.9	119.3	120.5	122.7	125.8	129.9	134.8	140.5
k=2	8.8	9.5	10.0	10.7	11.7	13.1	15.4	19.5	79.9	85.1	88.4	92.8	97.7	102.9	108.2	114.4
k=3	8.0	9.1	9.8	10.7	11.9	13.7	16.5	21.8	73.1	81.4	85.8	91.3	97.0	102.7	108.7	116.3
k=4	7.4	8.9	9.6	10.7	12.0	14.0	17.4	23.6	67.9	78.6	83.8	89.6	95.5	101.3	107.7	116.1

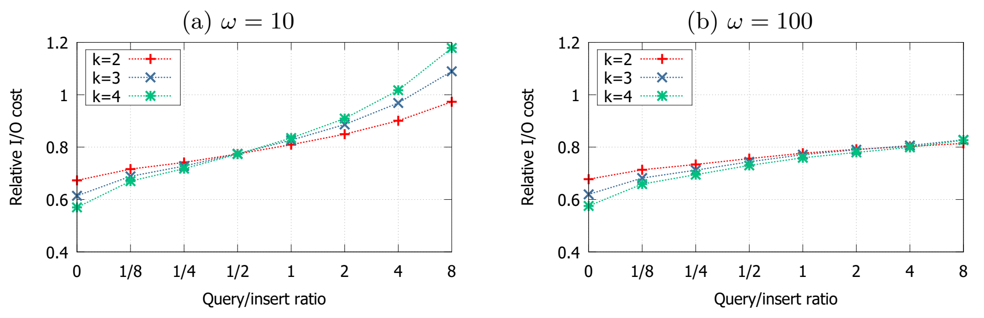


Figure 1 Relative I/O cost of k-level hash table with different values of k. The I/O costs are divided by the $k = 1$ case, so every data point below 1 indicates an improvement in such case. Numbers are from Table 2.

50% query keys are in the hash table (this ratio affects the I/O cost since a successful query can terminate earlier). The number of levels k varies from 1 to 4. In Table 2, we show the overall I/O costs, which are the weighted sums assuming two typical values of the write-read ratio ω, 10 and 100.

We first look at the number of write transfers. When there is no query (i.e., the first column, just inserting 1 million entries), the numbers of writes are consistent with our analysis for insertions in Section 4.1. The only exception here is that cache can hold a constant fraction of the elements, which batches the writes and reduces the number of memory transfers. However, the relative trend in each column remains unchanged. Namely, the number of writes always decreases with the increase of k regardless of the ratio between queries and updates. The number of writes is reduced by 33%, 40% and 43% when $k = 2, 3, 4$ respectively. Such improvement also shows up in the overall I/O cost in Table 2.

We note that more queries cause more reads, and larger k also leads to more reads. Since these reads flush the cache-lines, the numbers of writes in these cases also marginally increase. The optimal choice of k is decided by the update/query distribution as well as the write-read ratio ω. In general, more queries lead to worse performance with larger k, and larger ω prefers larger k. In Table 2, we underline the numbers indicating the best choice of k in that specific setting. The experiment results indicate that picking k to be 2 or 3 is always a good choice when $\omega = 10$, and 3 or 4 when $\omega = 100$.

4.3 Conclusions

We proposed a new data structure, the k-level hash table, to implement unordered set and map that has the same space utilization compared to the classic open-addressing hash tables. The key idea is to keep multiple instead of one level of hash tables. As a result, the algorithm uses fewer writes during resizings, at the cost of more reads in other operations.

The best choice of k is decided by the ratio of updates and queries. Our experiment shows that $k = 2$ always leads to a lower or similar I/O cost when the query/insert ratio is no more than 8, compared to the classic $k = 1$ setting. For the ratio of write/read cost is larger (like 100), larger values of k, like 3 or 4, are even more preferable than the $k = 2$ case.

5 Graph Traversal Algorithms

In this paper, we discuss two of the most commonly-used graph traversal algorithms: breadth-first search (BFS), and Dijkstra's algorithm. We show that using the new implementations discussed in this paper, these algorithms use much fewer writes in most cases, compared to the classic ones. Due to the page limit, we abstract our approaches and conclusions here in this section, and provide the full details in the full version of this paper [20].

Given a graph $G = (V, E)$, we assume $n = |V|$ is the number of vertices, and $m = |E|$ is the number of edges.

5.1 Breadth-First Search

We discuss our implementations and experiment results on breadth-first searches (BFS) on undirected graph traversing or searching. Our algorithms compute the single-source shortest paths (SSSP) or pairwise shortest-paths (given the specific source and target) on unweighted graphs, which can further apply to graph radii estimation, eccentricity estimation and betweenness centrality, and act as a basic building block for other graph algorithms like graph connectivity, reachability, biconnected components, and strongly connected components.

Implementations. The classic implementation of BFS keeps a vertex queue of size n, and an array of boolean flags of size n indicating whether each vertex is visited or not during the search. This implementation requires at most 2 writes per vertex, and the overall I/O cost of BFS $Q(n, m) = O(\omega n + m)$ [8]. This bound is asymptotically optimal for arbitrary graphs since the output size of BFS is $\Theta(n)$. However, a number of applications (e.g., s-t shortest-path or connectivity, graph radii estimation or eccentricity estimation) have output size $O(1)$, which allows utilizing the small-memory and reducing the number of writes.

The key observation to improve the write-efficiency is that, at any time, we only need the information of three consecutive frontiers (a frontier is the set of vertices with the same distance to the source node). We hence use the k-level hash table discussed in Section 4 to implement the frontiers. This avoids the writes to mark the visited flag of each vertex. We

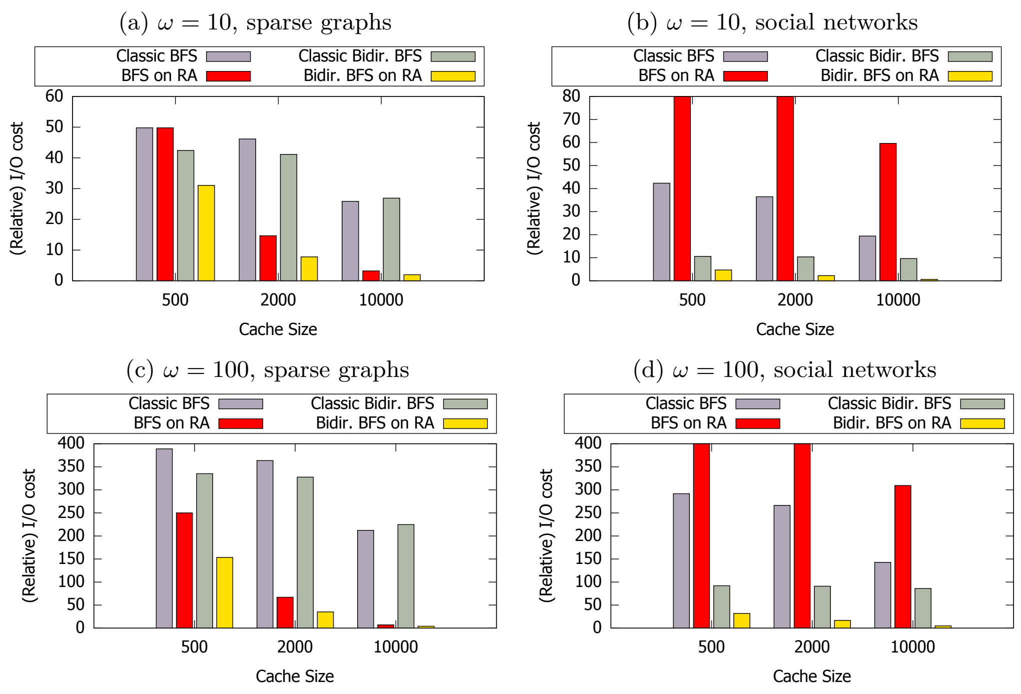


Figure 2 The trends of the I/O costs of four different implementations of BFS. The new implementations shown in this paper are the BFS and bidirectional BFS based on rotating arrays (red and yellow bars). The graphs used in the experiment are shown in full paper [20] and categorized into sparse (almost planar) graphs and social networks. We show the relative I/O cost based on varied cache sizes, and each number is geometric mean of the four graphs in that category. We can see the consistent advantages of the new BFS implementation on sparse graphs, and the improvement of the new bidirectional version in all cases. Notice that in (b) and (d) some values exceed the ranges of vertical axis.

note that once the frontier size fits into the small-memory, the algorithm does not require any writes to traverse the newly visited vertices. In the full paper [20] we show the average and maximum frontier sizes of the experiment graphs, which will help to understand the performance on these graph instances. To the best of our knowledge, we are unaware of any graph invariant to capture and predict the average and maximum frontier sizes, and we believe that it can be an interesting topic for further study. This algorithm is referred to as the *BFS on RA* (rotating arrays) in Figure 2, and more details of the implementation are given in the full paper [20].

The previous algorithm works well on graphs with larger diameters, but not on small-diameter graphs like social networks. We then introduce the bidirectional version (*Bidir. BFS on RA*) when the queries are s-t (pairwise) shortest paths, that overcomes the disadvantages of the previous algorithm. More analysis and details of this version are given in full version.

Experiment. Our experiment is based on eight graphs that are synthesized or from SNAP [29]. We show a significant improvement on all eight graphs with various cache sizes, compared to the classic queue-based implementations. Figure 2 is the bar charts that show the trends of the four implementations. When $\omega = 10$, our implementation shows an up-to 8-fold improvement on SSSP, and an up-to 43-fold improvement on s-t shortest-paths. For $\omega = 100$, the improvement is more stable, which is 69 on SSSP and 71 on s-t shortest-paths.

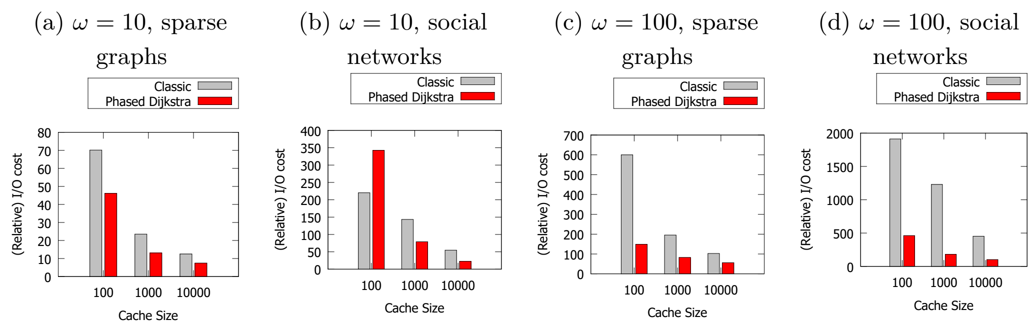


Figure 3 The trends of the I/O costs of classic Dijkstra (grey) and phased Dijkstra (red) on different graphs with varied cache sizes. The graphs used in the experiment are shown in the full paper [20] and categorized into sparse (almost planar) graphs and social networks. Each number of the I/O cost is geometric mean of the four graphs in that category. Phased Dijkstra performs consistently better in all cases except when both the cache size and the asymmetry ω are small.

Conclusions. We discuss how to efficiently implement BFS in the asymmetric setting and experiment the I/O performance for four implementations on a variety of undirected graphs. We show that for s-t (pairwise) distance queries, our bidirectional BFS using rotating arrays shows a significant advantage in all cases we tested. For single-source shortest-paths, the unidirectional BFS using rotating arrays has a significant improvement when the cache can hold every single frontier during the search.

5.2 Dijkstra's Algorithm

Dijkstra's Algorithm [14] computes single-source shortest paths on non-negative weighted graphs. The classic heap-based implementation requires $O(m \log(nB/M))$ reads and writes.

For the sake of write-efficiency, Blelloch et al. [8] discussed a variant called *Phased Dijkstra*. This algorithm only requires linear writes, but the algorithm is just explained at a high level and without much details. In this paper, we supplement the pseudocode, data structure design, and implementation details (in the full paper [20]).

Based on our implementation, we conduct various experiment to show the asymmetric I/O efficiency. In Figure 3, we show the trend of the relative I/O costs of the classic Dijkstra and phased Dijkstra with various cache sizes on different graphs. We show that phased Dijkstra outperforms classic Dijkstra in most cases except for the only case on social networks with very small cache size, and the improvement on I/O cost in all cases is up to 3 and 7.6 when ω is 10 and 100. We also consider various cache policies and sets of parameters and show that the improvement is consistent among all settings.

Summaries. We discuss phased Dijkstra and test its performance on a variety of graphs. The high-level idea is to fit the heap of Dijkstra's algorithm within the small-memory (i.e., the cache) and thus the algorithm applies no intermediate writes to the large asymmetry memory to maintain the heap. The extra cost is that, the algorithm is run in multiple phases each requiring $O(m)$ reads, but we show that such price is worthwhile in most cases. The experiments show that phased Dijkstra consistently outperforms the binary-heap version on I/O costs, except for the combination of small ω $(= 10)$, small cache size, and on social networks. Although phased Dijkstra contains several parameters, we also show that they can be chosen from a reasonably wide range and do not affect the superiority of phased Dijkstra. The same conclusion also holds for different cache policies.

We note that the idea of fitting the data structure or the computation in the small-memory can also be applied to computing minimum spanning tree, sorting, and many other problems.

References

1 Alok Aggarwal and Jeffrey S. Vitter. The Input/Output complexity of sorting and related problems. *Communications of the ACM*, 31(9), 1988. `doi:10.1145/48529.48535`.

2 Ameen Akel, Adrian M. Caulfield, Todor I. Mollov, Rajech K. Gupta, and Steven Swanson. Onyx: A prototype phase change memory storage array. In *USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2011.

3 Manos Athanassoulis, Bishwaranjan Bhattacharjee, Mustafa Canim, and Kenneth A. Ross. Path processing using solid state storage. In *International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS)*, 2012.

4 Avraham Ben-Aroya and Sivan Toledo. Competitive analysis of flash-memory algorithms. In *European Symposium on Algorithms (ESA)*, 2006.

5 Naama Ben-David, Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, Charles McGuffey, and Julian Shun. Parallel algorithms for asymmetric read-write costs. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2016.

6 Naama Ben-David, Guy E Blelloch, Jeremy T Fineman, Phillip B Gibbons, Yan Gu, Charles McGuffey, and Julian Shun. Implicit decomposition for write-efficient connectivity algorithms. In *Proc. IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2018.

7 Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, and Julian Shun. Sorting with asymmetric read and write costs. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2015.

8 Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, and Julian Shun. Efficient algorithms with asymmetric read and write costs. In *European Symposium on Algorithms (ESA)*, pages 14:1–14:18, 2016.

9 Guy E Blelloch, Phillip B Gibbons, Yan Gu, Charles McGuffey, and Julian Shun. The parallel persistent memory model. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2018.

10 Guy E Blelloch, Yan Gu, Julian Shun, and Yihan Sun. Parallel write-efficient algorithms and data structures for computational geometry. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, 2018.

11 Erin Carson, James Demmel, Laura Grigori, Nicholas Knight, Penporn Koanantakool, Oded Schwartz, and Harsha Vardhan Simhadri. Write-avoiding algorithms. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 648–658, 2016.

12 Shimin Chen, Phillip B. Gibbons, and Suman Nath. Rethinking database algorithms for phase change memory. In *Conference on Innovative Data Systems Research (CIDR)*, 2011.

13 Sangyeun Cho and Hyunjin Lee. Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009.

14 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1), 1959.

15 Xiangyu Dong, Norman P. Jouupi, and Yuan Xie. PCRAMsim: System-level performance, energy, and area modeling for phase-change RAM. In *ACM International Conference on Computer-Aided Design (ICCAD)*, 2009.

16 Xiangyu Dong, Xiaoxia Wu, Guangyu Sun, Yuan Xie, Hai H. Li, and Yiran Chen. Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement. In *ACM Design Automation Conference (DAC)*, 2008.

17 David Eppstein, Michael T. Goodrich, Michael Mitzenmacher, and Pawel Pszona. Wear minimization for cuckoo hashing: How not to throw a lot of eggs into one basket. In *ACM International Symposium on Experimental Algorithms (SEA)*, 2014.

18 Eran Gal and Sivan Toledo. Algorithms and data structures for flash memories. *ACM Computing Surveys*, 37(2), 2005.

19 Yan Gu. *Write-Efficient Algorithms (draft)*. PhD Thesis, 2018.

20 Yan Gu, Yihan Sun, and Guy E. Blelloch. Algorithmic building blocks for asymmetric memories (full version). In *arXiv preprint:1806.10370*, 2018.

21 HP, SanDisk partner on memristor, ReRAM technology. http://www.bit-tech.net/news/hardware/2015/10/09/hp-sandisk-reram-memristor, 2015.

22 Jingtong Hu, Qingfeng Zhuge, Chun Jason Xue, Wei-Che Tseng, Shouzhen Gu, and Edwin Sha. Scheduling to optimize cache utilization for non-volatile main memories. *IEEE Transactions on Computers*, 63(8), 2014.

23 www.slideshare.net/IBMZRL/theseus-pss-nvmw2014, 2014.

24 Intel and Micron produce breakthrough memory technology. http://newsroom.intel.com/community/intel_newsroom/blog/2015/07/28/intel-and-micron-produce-breakthrough-memory-technology, 2015.

25 Riko Jacob and Nodari Sitchinava. Lower bounds in the asymmetric external memory model. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 247–254, 2017.

26 Hyojun Kim, Sangeetha Seshadri, Clement L. Dickey, and Lawrence Chu. Evaluating phase change memory for enterprise storage systems: A study of caching and tiering approaches. In *USENIX Conference on File and Storage Technologies (FAST)*, 2014.

27 Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable DRAM alternative. In *ACM International Symposium on Computer Architecture (ISCA)*, 2009.

28 Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W Keller. Energy management for commercial servers. *Computer*, 36(12):39–48, 2003.

29 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection, 2014.

30 Jacob Leverich and Christos Kozyrakis. Reconciling high server utilization and sub-millisecond quality-of-service. In *European Conference on Computer Systems*, page 4. ACM, 2014.

31 Jasmina Malicevic, Subramanya Dulloor, Narayanan Sundaram, Nadathur Satish, Jeff Jackson, and Willy Zwaenepoel. Exploiting NVM in large-scale graph analytics. In *Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads*. ACM, 2015.

32 Krishna T Malladi, Ian Shaeffer, Liji Gopalakrishnan, David Lo, Benjamin C Lee, and Mark Horowitz. Rethinking DRAM power modes for energy proportionality. In *IEEE/ACM International Symposium on Microarchitecture*, pages 131–142, 2012.

33 Jagan S. Meena, Simon M. Sze, Umesh Chand, and Tseung-Yuan Tseng. Overview of emerging nonvolatile memory technologies. *Nanoscale Research Letters*, 9, 2014.

34 MARSSx86. http://marss86.org.

35 Hyoungmin Park and Kyuseok Shim. FAST: Flash-aware external sorting for mobile database systems. *Journal of Systems and Software*, 82(8), 2009. doi:10.1016/j.jss.2009.02.028.

36 PTLsim. http://www.ptlsim.org.

37 Moinuddin K. Qureshi, Sudhanva Gurumurthi, and Bipin Rajendran. *Phase Change Memory: From Devices to Systems*. Morgan & Claypool, 2011.

38 Daniel Sanchez and Christos Kozyrakis. Zsim: fast and accurate microarchitectural simulation of thousand-core systems. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 475–486. ACM, 2013.

39 Stratis D. Viglas. Adapting the B$^+$-tree for asymmetric I/O. In *East European Conference on Advances in Databases and Information Systems (ADBIS)*, 2012.

40 Stratis D. Viglas. Write-limited sorts and joins for persistent memory. *VLDB Endowment*, 7(5), 2014.

41 Cong Xu, Xiangyu Dong, Norman P. Jouppi, and Yuan Xie. Design implications of memristor-based RRAM cross-point structures. In *IEEE Design, Automation and Test in Europe (DATE)*, 2011.

42 Byung-Do Yang, Jae-Eun Lee, Jang-Su Kim, Junghyun Cho, Seung-Yun Lee, and Byoung-Gon Yu. A low power phase-change random access memory using a data-comparison write scheme. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2007.

43 Yole Developpement. Emerging non-volatile memory technologies, 2013.

44 Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. A durable and energy efficient main memory using phase change memory technology. In *ACM International Symposium on Computer Architecture (ISCA)*, 2009.

45 Omer Zilberberg, Shlomo Weiss, and Sivan Toledo. Phase-change memory: An architectural perspective. *ACM Computing Surveys*, 45(3), 2013.

On the Decision Tree Complexity of String Matching

Xiaoyu He
Institute of Computing Technology, Chinese Academy of Sciences, China, and
University of Chinese Academy of Sciences, China
hexiaoyu14@mails.ucas.ac.cn

Neng Huang
University of Chinese Academy of Sciences, China
huangneng14@mails.ucas.ac.cn

Xiaoming Sun
Institute of Computing Technology, Chinese Academy of Sciences, China, and
University of Chinese Academy of Sciences, China
sunxiaoming@ict.ac.cn

Abstract

String matching is one of the most fundamental problems in computer science. A natural problem is to determine the number of characters that need to be queried (i.e. the decision tree complexity) in a string in order to decide whether this string contains a certain pattern. Rivest showed that for every pattern p, in the worst case any deterministic algorithm needs to query at least $n - |p| + 1$ characters, where n is the length of the string and $|p|$ is the length of the pattern. He further conjectured that this bound is tight. By using the adversary method, Tuza disproved this conjecture and showed that more than one half of binary patterns are *evasive*, i.e. any algorithm needs to query all the characters (see Section 1.1 for more details).

In this paper, we give a query algorithm which settles the decision tree complexity of string matching except for a negligible fraction of patterns. Our algorithm shows that Tuza's criteria of evasive patterns are almost complete. Using the algebraic approach of Rivest and Vuillemin, we also give a new sufficient condition for the evasiveness of patterns, which is beyond Tuza's criteria. In addition, our result reveals an interesting connection to *Skolem's Problem* in mathematics.

2012 ACM Subject Classification Theory of computation → Oracles and decision trees

Keywords and phrases String Matching, Decision Tree Complexity, Boolean Function, Algebraic Method

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.45

Related Version Full version is available at `https://arxiv.org/abs/1712.09738`

Funding This work was supported in part by the National Natural Science Foundation of China Grant 61433014, 61502449, 61602440, and the 973 Program of China Grants No.2016YFB1000201.

1 Introduction

The string matching problem is one of the most fundamental problems in computer science. The goal of string matching problem is to find one or all occurrences of a pattern in an input string. Lots of efficient algorithms have been discovered in the 20th century. For example, the KMP algorithm [7], discovered by Knuth, Morris and Pratt, is able to locate all occurrences of a pattern of length m in a string of length n in $O(n + m)$ time. This is essentially the best

© Xiaoyu He, Neng Huang, and Xiaoming Sun;
licensed under Creative Commons License CC-BY

possible since every algorithm needs $\Omega(n+m)$ time to process the input. Another elegant algorithm is the Karp-Rabin algorithm [6], which uses hashing and can be used to search for a set of patterns. A detailed treatment of these algorithms can be found in [3]. However, the problem becomes subtler when we adopt a different complexity measure, which is the number of characters that the algorithm has to examine in the input string given the prior knowledge of the pattern string. When we confine the alphabet to $\{0,1\}$, this measure is exactly the *decision tree complexity* of boolean string matching problem. Recall that for a binary function f, its decision tree complexity is the number of bits that we have to examine in the worst case in any input x in order to compute $f(x)$.

1.1 Notations and Previous Work

Let p be a pattern over alphabet Σ with $|\Sigma| = \sigma$. Throughout the paper, let $p[i]$ be the i-th character in p and $p[i..j]$ be the substring of p indexed from i to j. Let A_p be a deterministic string searching algorithm which searches for p in any given string s. Following Rivest [9], we denote $w(A_p, n)$ to be the maximum number of characters that A_p examines for any s of length n. Let $D_p(n) = \min_{A_p} w(A_p, n)$, where A_p is taken over all deterministic string searching algorithms. When the alphabet $\Sigma = \{0,1\}$, D_p is exactly the boolean decision tree complexity of string searching algorithm with pattern p. It is clear that this function is monotone, since we can simply add some redundant characters at the end of the searched text. We state this as the following proposition.

▶ **Proposition 1.** *For every pattern p and $n \in \mathbb{N}$, $D_p(n) \leq D_p(n+1)$.*

We define the evasiveness of a pattern as follows.

▶ **Definition 2.** A pattern p is called *evasive* if there exists $N_0 \in \mathbb{N}$ such that for all $n > N_0$, $D_p(n) = n$.

By this definition, a pattern p is evasive if for every algorithm A and every sufficiently large n, there is a string s of length n such that A has to query every character of s in order to determine whether s contains p as a substring. We are interested in determining what patterns are evasive and what patterns are not.

Let $|p|$ denote the length of a pattern p. Rivest gave the following linear lower bound on $D_p(n)$:

▶ **Theorem 3** ([9]). *For every pattern p, $D_p(n) \geq n - |p| + 1$ for all $n \in \mathbb{N}$.*

To prove this theorem, Rivest showed that for every $n \in \mathbb{N}$, there exists an integer i between 0 and $|p|$ such that $D_p(n+i) = n+i$, then combined with Proposition 1, Theorem 3 follows. Based on this result, we define *non-evasiveness* as follows.

▶ **Definition 4.** A pattern p is called *non-evasive* if for every $N_0 \in \mathbb{N}$ there exists $n > N_0$ such that $D_p(n) = n - |p| + 1$.

As discussed above, what Rivest proved in fact implies that it is impossible for a pattern to achieve the lower bound in Theorem 3 on consecutive integers, which is the reason we define non-evasiveness in this way. Rivest showed that the pattern $p = 1^k$ (and therefore 0^k) is non-evasive. He further conjectured that all patterns are non-evasive. However, this conjecture was later disproved by Tuza in [11]. We briefly summarize Tuza's work here. Given a string b, let $BE(b)$ denote the set of patterns prefixed and suffixed by b, but other than b. Also, for patterns u and v, let uv denote their concatenation. If $p \in BE(b)$, then let $p(b)$ be the string ubv where, $ub = bv = p$. Tuza proved the following result.

▶ **Theorem 5** ([11]). *Let $p \in BE(b)$. If*
1. *$p(b)$ does not contain a substring p' of length $|p|$ other than prefix or suffix of $p(b)$ such that p' and p differ from each other in at most two characters, and*
2. *the pattern pp does not contain a substring p' of length $|p|$ other than prefix or suffix of pp such that p' and p differ from each other in at most four characters,*

and $n \geq |p|(2|p| - |b|)/\gcd(|p|, |b|)$, then $D_p(n) \geq n - k$, where $k = n \mod \gcd(|p|, |b|)$.

If a pattern string p satisfies the conditions in Theorem 5 and $\gcd(|p|, |b|) = 1$, then one would have $D_p(n) = n$ for all sufficiently large n. This implies that p is evasive and therefore serves as a counterexample of Rivest's conjecture. Tuza estimated the proportion of pattern strings which satisfy the conditions in Theorem 5 and proved that when $\Sigma = \{0, 1\}$, there exists more than $0.5061 \cdot 2^m$ evasive patterns of length m.

Beyond the worst case complexity, the average-case complexity has also been studied previously, that is, finding out the numbers of characters that need to be examined on average assuming that the input string is sampled from the uniform distribution. Yao [12] showed that, for almost all patterns of large enough length m, an algorithm needs to examine $\Theta(\frac{n \log_q m}{m})$ characters on a uniformly random input string of length $n > 2m$, here q is the size of the alphabet.

1.2 Our Contributions

In this paper we settle the decision tree complexity for almost every string except an $o(1)$ fraction. More precisely, we prove that Tuza's lower bound, which is developed combinatorially, is in fact tight for almost every string, by showing an algorithm which achieves this lower bound. This algorithm is based on the periods of the pattern string.

▶ **Definition 6** (Periods). Let p be a pattern of length m and k be a positive integer no larger than m. We say that p is k-periodic, or p has a period k, if $p[i] = p[i + k]$ for all $1 \leq i \leq n - k$. Let $\mathbf{Period}(p) = \{k | p \text{ } is \text{ } k\text{-periodic}\}$ be the set of all periods of p.

The definition here is the same as in [4], in which it was used to develop a time-space optimal algorithm. A similar idea can also be found in [11]. For set $S \subset \mathbb{N}$, let $\gcd(S)$ denote the greatest common divisor of all elements in S. Here is our main theorem.

▶ **Theorem 7** (Main). *Let p be a pattern of length m and $c = \gcd(\mathbf{Period}(p))$ be the greatest common divisor of all p's periods, then $D_p(n) = n - (n \mod c)$, except for an $O(m^5 \sigma^{-m/2})$ fraction of patterns.*

Here, the fraction of patterns is computed in the following way. We first fix a pattern length m, and then count the number of patterns of length m that satisfy some certain properties, then compute its ratio to the total number of length-m patterns, which is σ^m. We then investigate the asymptotic behavior of this ratio as m goes to infinity.

By Theorem 7, the fraction of patterns whose decision tree complexity we don't know goes to 0 as the pattern length goes to infinity.

Besides this result, we also use the algebraic approach to show the evasiveness of certain family of patterns, for which Tuza's method does not work. This algebraic approach was first developed by Rivest and Vuillemin [10], and we extend it to our problem. Interestingly, we find that this approach reveals a relation between our problem and the *Skolem's Problem*. We also define the *characteristic polynomial* of a pattern, which is again closely related to the pattern's periodic behaviors. This polynomial, besides its application in this problem, is of independent interest on its own.

2 Upper Bounds

In this section, we prove one direction of Theorem 7, which can be stated as the following lemma.

▶ **Lemma 8.** *Let p be a pattern of length m and $c = \gcd(\mathbf{Period}(p))$ be the greatest common divisor of all p's periods, then $D_p(n) \le n - (n \mod c)$.*

To show this lemma, we will develop an algorithm whose behavior depends on the periods of the pattern string.

2.1 Non-evasiveness of Bifix-free Patterns

We first look at the simple case where our pattern is bifix-free.

▶ **Definition 9.** A string s is called a *bifix* of a string t if s is both a prefix and a suffix of t. A pattern p is called *bifix-free* if p has no bifix other than itself.

▶ Remark (Relations to combinatorics on words). The concepts of periods and bifixes are also studied in the field of combinatorics on words under possibly different names. Bifixes are usually referred to as *borders* in combinatorics on words, and bifix-free strings are usually called *unbordered words*. For more details from viewpoint of combinatorics on words, see [2].

Bifix-free patterns have the following property in terms of periods.

▶ **Lemma 10.** *A pattern p of length n has a bifix of length $k < n$ if and only if it is $(n-k)$-periodic. Furthermore, p is bifix-free if and only if it has only one period, which is n.*

Proof. If a pattern p has a period $k < n$, then $p[1..(n-k)] = p[(k+1)..n]$. This is equivalent to say that p has a prefix of length $n-k$ which is equal to p's suffix of length $n-k$. The "furthermore" part follows directly. ◀

Then, for a bifix-free pattern p we have $|p| = \gcd(\mathbf{Period}(p))$. According to Lemma 8, the following result is expected.

▶ **Lemma 11.** *Let p be a bifix-free pattern of length m, then $D_p(n) \le n - (n \mod m)$ and p is non-evasive.*

Proof. Consider the algorithm in Figure 1. We claim that this algorithm can produce the correct output after $n - (n \mod m)$ queries to the string. Suppose that in Line 11, we find that $s[i..j]$ is not equal to p, otherwise we can stop and output this occurrence. Note that until Line 11 we have only queried m characters in s, which are $s[i], s[i+1], \dots, s[j]$. We show that for indices l with $1 \le l \le m$, we have $s[l..(l+m-1)] \ne p$.

- $1 \le l < i$. In this case, there exists an index t with $i \le t \le l+m-1$ such that $s[t..(l+m-1)]$ is not a suffix of p, since otherwise $s[l+m]$ would not be queried, contradicting the fact that $j = i+m-1 \ge l+m$. And therefore we have $s[l..(l+m-1)]$ is not suffix of p.
- $l = i$. In this case we have by assumption that $s[l..(l+m-1)] = s[i..j] \ne p$.
- $i < l \le m$. Assume that $s[l..(l+m-1)]$ equals to p. Then for all indices t with $l \le t \le j$, $s[l..t]$ is a prefix of p, and therefore by bifix-freeness, is not a suffix of p. However, since $i < l$, $s[l-1]$ is queried, so there must exists such an index t that $s[l..t]$ is a suffix of p, which is a contradiction. Hence $s[l..(l+m-1)]$ does not equal to p.

```
Input: string $s$ of length $n$, bifix-free pattern $p$ of length $m$
Output: whether $p$ is a substring of $s$
 1: function FIND($s$, $p$)
 2:     if $n < m$ then
 3:         return $false$
 4:     $i \leftarrow m$, $j \leftarrow m$
 5:     query($s[m]$)
 6:     while $j - i \neq m - 1$ do
 7:         if $s[i..j]$ is a suffix of $p$ then
 8:             query($s[i-1]$), $i \leftarrow i - 1$
 9:         else
10:             query($s[j+1]$), $j \leftarrow j + 1$
11:     if $s[i..j] = p$ then
12:         return $true$
13:     else
14:         return FIND($s[m+1..n]$, $p$)
15: end function
```

Figure 1 Algorithm for bifix-free patterns.

Table 1 The table for the first three significant digits for b_∞^σ when $\sigma \leq 6$.

σ	2	3	4	5	6
b_∞^σ	0.268	0.557	0.688	0.760	0.801

This shows that after querying m characters, we either find an occurrence of p in s, or reduce the size of s by m. When the size of s is smaller than m, the algorithm trivially stops. Therefore after $n - (n \mod m)$ queries, we will be able to determine whether s contains p. This establishes an upper bound on $D_p(n)$, namely $D_p(n) \leq n - (n \mod m)$, which matches Rivest's lower bound. We conclude that bifix-free patterns are non-evasive. ◀

We note that the above algorithm is in fact applicable for all finite alphabets. For an alphabet Σ of size σ, we define b_m^σ to be the proportion of bifix-free strings in strings of length m, that is,

$$b_m^\sigma = \frac{|\{p \in \Sigma^m | p \text{ is bifix-free}\}|}{\sigma^m}, |\Sigma| = \sigma.$$

Nielsen [8] showed that the sequence $\{b_m^\sigma\}_{m=1}^\infty$ converges. Furthermore, he proved that

$$b_\infty^\sigma := \lim_{m \to \infty} b_m^\sigma \geq 1 - \sigma^{-1} - \sigma^{-2}.$$

Table 1 (from [8]) shows the first three significant digits for b_∞^σ when $\sigma \leq 6$.

From this we obtain that more than 26.7% of binary pattern strings of length m are non-evasive, where m is sufficiently large. We also note that, as the size of the alphabet increases, the percentage of patterns that are non-evasive tends to 1.

Algorithm 1 Algorithm for general patterns.

Input: string s of length n, pattern p of length m
Output: whether p is a substring of s

```
1:  function FIND($s$, $p$)
2:      if $n < m$ then return $false$
3:      $i \leftarrow m$, $j \leftarrow m$
4:      query($s[m]$)
5:      while $j - i \neq m - 1$ do
6:          if $s[i..j]$ is a suffix of $p$ then
7:              query($s[i-1]$), $i \leftarrow i - 1$
8:          else
9:              query($s[j+1]$), $j \leftarrow j + 1$
10:     if $s[i..j] = p$ then
11:         return $true$
12:     $l \leftarrow m + c$
13:     while $l \leq n$ do
14:         $i \leftarrow l$, $j \leftarrow l$
15:         query($s[l]$)
16:         repeat
17:             if $s[i..j]$ is a suffix of $p$ then
18:                 query($s[i-1]$), $i \leftarrow i - 1$
19:             else
20:                 query($s[j+1]$), $j \leftarrow j + 1$
21:         until $c$ new characters have been queried OR $j - i = m - 1$
22:         if $s[(j-m+1)..j] = p$ then
23:             return $true$
24:         $l \leftarrow l + c$
25:     return $false$
26: end function
```

2.2 The General Case

In the previous section, we used bifix-freeness as a crucial tool in our algorithm. The property stated in Lemma 10 is in fact playing an important role here. It is natural to ask that what if a pattern has periods other than its own length? An intuition is that if a pattern has good periodic behaviors, then a well-behaved algorithm must exist as well. We therefore formalize this intuition and give the proof of Lemma 8.

Proof of Lemma 8. Let's consider the algorithm in Algorithm 1, which is a generalization of the algorithm for bifix-free patterns. Intuitively, this algorithm examines the string by blocks of size c, which is the greatest common divisor of p's periods. Note that for simplicity we formulate this algorithm in a way that it may query the same character more than once. In such cases, we can reuse the previous result and need not really query that character. Our algorithm might also query a character in s with index larger than n. In such cases, we assume that we obtain a character different than $p[m]$, such that it cannot form the pattern p with previous characters. We also assume that $c > 1$.

First of all, it is easy to see that this algorithm queries at most $n - (n \mod c)$ characters in s. We now show that this algorithm returns the answer correctly. Our algorithm only returns *true* when it really see the pattern p, so it suffices to show that if there are occurrences of p in s, then our algorithm will always be able to find one. Here we prove that it will always find the first occurrence.

Assume that the first occurrence of p in s is $s[k-m+1..k]$ and $k = hc + t$ for some $0 \le t < c$. We want to show that, when our algorithm starts to examine the (hc)-th character of the string at Line 15 (it could be that our algorithm will be able to locate p in the while loop beginning at Line 5, but that case is even simpler), there are at most c characters in $s[k-m+1..k]$ which have not been queried yet. If this holds, then our algorithm will be able to identify $s[k-m+1..k]$ as p in at most c queries.

In fact, we prove a strong claim that whenever $k-m < l \le k$, in order for the repeat-until loop at Line 16-21 to stop, we either either query c new characters in the range $s[k-m+1..k]$, or we have queried every character in the range $s[k-m+1..l+t]$.

Suppose our algorithm is going to query a character with index smaller than $k-m+1$ when $k-m < l \le k$, then at some point our algorithm will query $s[k-m]$ at Line 18. Clearly, $i = k-m+1$ at that moment. Also, it must be that $j = l+t$, for when Line 18 is executed, $s[i..j]$ must be a suffix of p. But we also know that $s[i..j]$ is a prefix of p. Thus the length of $s[i..j]$, which is $j-k+m$, must be a multiple of c, implying that $j = l+t$ (since l is always a multiple of c). If our algorithm do not query a character with index smaller than $k-m+1$ when $k-m < l \le k$, then in order for the repeat-until loop at Line 16 to end, we have all our c new characters in $s[k-m+1..k]$. This proves what we need, and the correctness of our algorithm follows. ◀

3 Proof of Theorem 7

In this section, we give the proof of our main theorem. We have proved one direction in Section 2. For the other direction, we use a similar analysis to Tuza's in [11]. We first restate (a stronger version of) Tuza's theorem here.

▶ **Theorem 12** ([11]). *Assume that $p \in BE(b_1), p \in BE(b_2), \ldots, p \in BE(b_l)$. If*

1. *for every $1 \le i \le l$, $p(b_i)$ does not contain a substring p' of length $|p|$ other than prefix or suffix of $p(b_i)$ such that p' and p differ from each other in at most two characters, and*
2. *the pattern pp does not contain a substring p' of length $|p|$ other than prefix or suffix of pp such that p' and p differ from each other in at most four characters,*

then for sufficiently large n, $D_p(n) \ge n-k$, where $k = n \mod \gcd(\{|p|, |b_1|, |b_2|, \ldots, |b_l|\})$.

We note that in Tuza's language, $p \in BE(b)$ essentially means that p has a bifix b. As is shown in Lemma 10, it is equivalent to say that p is $(|p|-|b|)$-periodic. Thus the condition $p \in BE(b_1), p \in BE(b_2), \ldots, p \in BE(b_l)$ is simply saying that p has periods $|p|-|b_1|, |p|-|b_2|, \ldots, |p|-|b_l|$, other than its own length $|p|$, and the expression $\gcd(\{|p|, |b_1|, |b_2|, \ldots, |b_l|\}$ is equivalent to $\gcd(\mathbf{Period}(p))$. To prove Theorem 7, we need the following two lemmas. These two lemmas are generalizations of Lemma 11 and Lemma 12 in [11].

▶ **Lemma 13.** *Let $B_1(n)$ be the set of patterns p such that $|p| = n$, $p \in BE(b)$ for some b and $p(b)$ contains a substring p' of length n other than prefix or suffix of $p(b)$ such that p' and p differ from each other in at most two characters. Then $|B_1(n)| = O(n^4\sigma^{n/2})$.*

▶ **Lemma 14.** *Let $B_2(n)$ be the set of patterns p such that $|p| = n$ and the pattern pp contains a substring p' of length n other than prefix or suffix of pp such that p' and p differ from each other in at most four characters. Then $|B_2(n)| = O(n^5\sigma^{n/2})$.*

Proof of Theorem 7. Let p be a pattern of length m. If $p \notin B_1(m) \cup B_2(m)$, then by Theorem 12, $D_p(n) \geq n - k$, where $k = n \mod \gcd(\mathbf{Period}(p))$. Also, by Lemma 8, $D_p(n) \leq n - k$. Therefore $D_p(n) = n - k$ for all $p \notin B_1(m) \cup B_2(m)$. By Lemma 13 and Lemma 14, $|B_1(m) \cup B_2(m)| = O(m^5 \sigma^{m/2})$, and hence Theorem 7 follows. ◀

For simplicity, from now on we say $p(b)$ *has property 1* if $p \in BE(b)$ and $p(b)$ contains a substring p' of length n other than prefix or suffix of $p(b)$ such that p' and p differ from each other in at most two characters, and we say p *has property 2* if the pattern pp contains a substring p' of length n other than prefix or suffix of pp such that p' and p differ from each other in at most four characters.

3.1 Proofs of the Two Lemmas

Now we prove Lemma 13 and Lemma 14. Tuza proved the case when $\Sigma = \{0, 1\}$ in [11]. We will adapt his proof to handle the case where Σ is any finite alphabet.

▶ **Lemma 15.** *If $p(b)$ has property 1 for some $|b| > |p|/2$, then we can find b' with length at most $|p|/2$ such that $p(b')$ has property 1 as well.*

Proof. If $p \in BE(b)$ for some $|b| > |p|/2$, then by definition, $p[i] = p[i + |p| - |b|]$ for every $1 \leq i \leq |b|$. Assume that $k(|p| - |b|) < |p| \leq (k + 1)(|p| - |b|)$ for some k, then let $b' = p[k(|p| - |b|) + 1..|p|]$. It is straightforward to check that b' is also a bifix of p and $p(b')$ contains $p(b)$ as a substring. Therefore $p(b')$ has property 1 if $p(b)$ has property 1. ◀

Proof of Lemma 13. Let $p \in B_1(n)$. By definition and Lemma 15, for some $|b| \leq |p|/2$, $p(b)$ contains a substring p' of length n other than prefix or suffix of $p(b)$ such that p' and p differ from each other in at most two characters. These at most two characters can be chosen in $(\sigma - 1)^2 n(n - 1)/2 + (\sigma - 1)n + 1$ different ways. Assume that p' starts in the $(i + 1)$-th character in $p(b)$, then after we fix these two erroneous locations, the first $\gcd(i, n - |b|)$ characters in $p(b)$ will uniquely determine $p(b)$. Therefore we have

$$\begin{aligned}
|B_1(n)| &\leq \sum_{|b|=1}^{n/2} \sum_{i=1}^{n-|b|-1} ((\sigma - 1)^2 n(n - 1)/2 + (\sigma - 1)n + 1)\sigma^{\gcd(i, n-|b|)} \\
&\leq n^2\sigma^2 \sum_{|b|=1}^{n/2} \sum_{i=1}^{n-|b|-1} \sigma^{\gcd(i, n-|b|)} \\
&\leq n^2\sigma^2 \cdot \frac{n}{2} \cdot n\sigma^{n/2} \\
&= \frac{n^4}{2}\sigma^{n/2+2}.
\end{aligned}$$ ◀

Proof of Lemma 14. The proof is similar to that of Lemma 13. Let $p \in B_2(n)$. Then the pattern pp has a substring p' that differs from p in at most four characters. These at most four characters can be chosen in at most $|\Sigma|^4 n^4$ different ways. Assume that p' starts in the $(i + 1)$-th position, then the first $\gcd(i, n)$ characters in p uniquely determines p. Therefore we have

$$|B_2(n)| \leq \sum_{i=1}^{n-1} \sigma^4 n^4 \sigma^{\gcd(i,n)} \leq \sum_{i=1}^{n-1} \sigma^4 n^4 \sigma^{n/2} \leq n^5 \sigma^{n/2+4}.$$ ◀

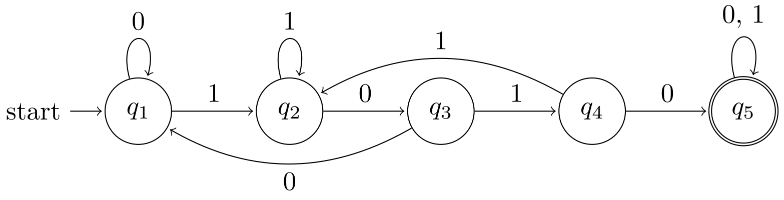


Figure 2 The finite state automaton for pattern $p = 1010$.

4 A Sufficient Condition for Evasiveness

Now that we have finished the proof of Theorem 7, a natural question to ask is what patterns lie outside the scope of Theorem 7? Rivest has given an example in [9], by showing that the pattern 1^n is non-evasive while $\gcd(\mathbf{Period}(1^n)) = 1$ since every integer between 1 and n is a period of 1^n. In this section, we will use the algebraic method to develop a new sufficient condition for evasiveness. We will assume that the alphabet $\Sigma = \{0, 1\}$. We first introduce the notion of characteristic polynomial in Section 4.1 and then state our theorem in Section 4.2. In Section 4.3, we will show the relationship between a pattern's periods and its characteristic polynomial, which allows for a convenient way to calculate the polynomial.

4.1 The KMP Automaton and the Transition Matrix

Following Rivest [9] we will make use of the finite state automaton constructed by the Knuth-Morris-Pratt algorithm. Let p be a pattern string of length m, then the automaton constructed will have $m+1$ states, where state q_1 is the initial state and state q_{m+1} is the only accepting state. The automaton reaches state q_i if the previous $i-1$ characters is a prefix of p where i is the largest possible among such ones, and the pattern p is not found already. The automation reaches state q_{m+1} as soon as the pattern p is found, and stays there forever. See Figure 2 for an example of the KMP automaton when the pattern $p = 1010$.

Let $U_p(n, i)$ be the set of strings of length n on which the automaton ends in state q_i. Let $g_p(n, i) := \sum_{s \in U_p(n,i)} x^{wt(s)}$, where $wt(s)$ is the number of 1's in s. The following lemma is used in [10] to show evasiveness of boolean functions.

▶ **Lemma 16** ([10]). *If $D_p(n) \le n - l$ for some integer $1 \le l \le n$, then $(x+1)^l$ divides $g_p(n, m+1)$.*

A useful consequence of this lemma is the following corollary.

▶ **Corollary 17.** *If there exists $N_0 \in \mathbb{N}$ such that $g_p(n, m+1) \not\equiv 0 \mod (x+1)$ for all $n > N_0$, then $D_p(n) = n$, i.e. p is evasive.*

By Lemma 16, we are only interested in the value of $g_p(n, m+1)$ modulo $x+1$. Note that we always have

$$g_p(n+1, m+1) = (x+1) g_p(n, m+1) + y \cdot g_p(n, m),$$

where y equals 1 or x depending on the last bit of the pattern string. Taking modulus of $(x+1)$ on both sides, we obtain

$$g_p(n+1, m+1) \equiv y \cdot g_p(n, m) \mod (x+1).$$

Since $y \equiv \pm 1 \mod (x+1)$ (with the sign determined by the last bit of the pattern string), we obtain the following lemma.

▶ **Lemma 18.** *$g_p(n+1, m+1) \equiv 0 \mod (x+1)$ if and only if $g_p(n,m) \equiv 0 \mod (x+1)$. Moreover, if there exists $N_0 \in \mathbb{N}$ such that $g_p(n,m) \not\equiv 0 \mod (x+1)$ for all $n > N_0$, then p is evasive.*

Now we define the transition matrix. Given a pattern string p of length m, we can express $g_p(n+1,1), \dots, g_p(n+1,m)$ in terms of $g_p(n,1), \dots, g_p(n,m)$. For example, when $p = 1010$, according to the automata in Figure 2, we can write

$$\begin{pmatrix} g_p(n+1,1) \\ g_p(n+1,2) \\ g_p(n+1,3) \\ g_p(n+1,4) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ x & x & 0 & x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & x & 0 \end{pmatrix} \begin{pmatrix} g_p(n,1) \\ g_p(n,2) \\ g_p(n,3) \\ g_p(n,4) \end{pmatrix}$$

Since we are only interested in these values modulo $x+1$, we may plug in $x = -1$ into all these terms. We denote $\overline{g}_p(n,i)$ to be the value obtained by plugging $x = -1$ into $g_p(n,i)$. In the previous example where $p = 1010$, we will obtain

$$\begin{pmatrix} \overline{g}_p(n+1,1) \\ \overline{g}_p(n+1,2) \\ \overline{g}_p(n+1,3) \\ \overline{g}_p(n+1,4) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ -1 & -1 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} \overline{g}_p(n,1) \\ \overline{g}_p(n,2) \\ \overline{g}_p(n,3) \\ \overline{g}_p(n,4) \end{pmatrix}$$

We call the matrix on the right hand side of the above equation the *transition matrix* of the pattern string $p = 1010$. In general, given a pattern string p, we write down the recurrence relation of $g_p(n,i)$ in the matrix form and plug in $x = -1$, and the resulting matrix will be our transition matrix. Let T_p denote the transition matrix for pattern p.

We will see later that the eigenvalues of T_p are of great use to us. We establish the following lemma using the characteristic polynomial of T_p. In the remaining part of this paper we will refer to the characteristic polynomial of T_p as characteristic polynomial of the pattern p.

▶ **Lemma 19.** *Let p be a pattern of length m. Let $P(\lambda) = \lambda^m + c_{m-1}\lambda^{m-1} + \cdots + c_0$ be the characteristic polynomial of p, then we have the recurrence relation*

$$\overline{g}_p(n+m,m) + c_{m-1}\overline{g}_p(n+m-1,m) + \cdots + c_0\overline{g}_p(n,m) = 0. \tag{1}$$

Proof. By the Cayley-Hamilton theorem (see Theorem 5.2.3 in [1]), we have

$$T_p^m + c_{m-1}T_p^{m-1} + \cdots + c_0 I = 0,$$

where I is the identity matrix. Right multiply both sides by column vector

$$\overline{g}_p(n) = (\overline{g}_p(n,1), \overline{g}_p(n,2), \dots, \overline{g}_p(n,m))^t,$$

we obtain

$$\overline{g}_p(n+m) + c_{m-1}\overline{g}_p(n+m-1) + \cdots + c_0\overline{g}_p(n) = 0,$$

since $T_p\overline{g}_p(n) = \overline{g}_p(n+1)$. Both sides of the equation above are m-dimensional column vectors, and we get the desired recurrence relation by looking at the last row of both vectors. ◀

4.2 The Skolem Problem and Finite Zeroes

Lemma 19 gives us a tool to get around $\overline{g}_p(n,1),\ldots,\overline{g}_p(n,m-1)$ and focus only on $\overline{g}_p(n,m)$. Now we are faced with the following problem:

> Let $\{u_n\}$ be a linear recurrent sequence. Does there exist N_0 such that u_n is non-zero for all $n > N_0$?

This problem is very similar to the *Skolem's Problem*, which can be stated as follows:

> Let $\{u_n\}$ be a linear recurrent sequence. Does there exist n such that $u_n = 0$?

For a detailed survey of the Skolem's problem, readers are referred to [5]. We will use the following result from [5], which partially solved our problem.

▶ **Lemma 20** ([5]). *Assume sequence $\{u_n\}_{n=1}^{\infty}$ satisfies*

$$u_n = a_{m-1}u_{n-1} + \cdots + a_1u_{n-m+1} + a_0u_{n-m},$$

where $a_0, a_1, \ldots, a_{m-1}$ are fixed integers. Also assume that $p(\lambda) = \lambda^m - a_{m-1}\lambda^{m-1} - \cdots - a_1\lambda - a_0$ has the decomposition

$$p(\lambda) = (\lambda-\lambda_1)^{m_1}(\lambda-\lambda_2)^{m_2}\cdots(\lambda-\lambda_r)^{m_r},$$

where $\lambda_1,\ldots,\lambda_r \in \mathbb{C}$ are distinct roots of $p(\lambda)$ and $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_r|$. Then there exists $N_0 \in \mathbb{N}$ such that u_n is non-zero for all $n > N_0$ if one of the following cases holds:

1. $|\lambda_1| > |\lambda_2|$.
2. $|\lambda_1| = |\lambda_2| > |\lambda_3|,\ \lambda_1 = \overline{\lambda_2}$.
3. $|\lambda_1| = |\lambda_2| = |\lambda_3| > |\lambda_4|,\ \lambda_1 \in \mathbb{R},\ \lambda_2 = \overline{\lambda_3}$.

The proof of this lemma can be found in Proposition 4.1 in [5]. Note that our statement is a little bit different. In [5] it is proved that the Skolem's problem is decidable in these cases, by showing that there exists an algorithmically computable constant N_0 such that $u_n \neq 0$ for all $n \geq N_0$, and therefore an algorithm for deciding the Skolem's problem only needs to check whether there are zeroes below the bound N_0.

Using this lemma, we can show the evasiveness of some pattern strings. As an example, we prove the following proposition.

▶ **Proposition 21.** *The pattern $p = 10^k1$ is evasive when $k > 0$.*

Proof. To begin with, we calculate its characteristic polynomial, which is

$$p(\lambda) = \det(\lambda I - T_p) = \lambda^{k+2} - \lambda + 1.$$

We note that this is also the characteristic polynomial for the recurrence of $\overline{g}(n,k+2)$ (here $|p| = k+2$). Now assume $z = re^{i\theta}$ is a root of $p(\lambda) = 0$. Then we have $|z^{k+2}| = |z-1|$, which implies

$$r^{k+2} = \sqrt{r^2 + 1 - 2r\cos\theta}.$$

This shows that for every r the value of $\cos\theta$ is determined, and therefore there can be at most 2 choices of θ. Thus, the pattern p either satisfies condition 1 or condition 2 in Lemma 20. We conclude that p is evasive. ◀

▶ Remark. The evasiveness of pattern $p = 10^k1$ is not covered by Tuza's Theorem. Though $p \in BE(b)$ where $b = 1$, the pattern 10^k110^k1 has a substring $p' = 0^k11$ which differs from p in only two positions, and thus violates the condition (b) in Theorem 5.

4.3 The Characteristic Polynomial and Periods

Writing down the characteristic polynomial through the transition matrix can sometimes be inefficient. Here we develop a faster way to calculate a pattern's characteristic polynomial and show some interesting connection to the periodic behavior of the pattern.

We give the following formula for a pattern's characteristic polynomial in terms of the pattern's periods. The proofs of results in this subsection can be found in the full version of this paper.

▶ **Theorem 22.** *Let p be a pattern of length m. Let $P(\lambda)$ be the characteristic polynomial of p, then we have $P(\lambda) = \lambda^m + c_{m-1}\lambda^{m-1} + \cdots + c_1\lambda + c_0$, where for $1 \le k \le m$,*

$$c_{m-k} = \begin{cases} (-1)^{wt(p[1..k])}, & \text{if } k \text{ is a period of } p, \\ 0, & \text{otherwise.} \end{cases}$$

In proving the above theorem, the following two lemmas will be useful.

▶ **Lemma 23.** *Let p be a pattern of length m. Assume that state q_m of the KMP automaton for p has a transition back to state q_{m-k+1} where $k \le m$.*

- *If $k = m$, then all patterns of $p[1..m-1]$ are preserved. That is to say, if $p[1..m-1]$ is l-periodic for some l, then $p[1..m]$ is also l-periodic.*
- *If $k < m$, then k is the smallest of the periods of $p[1..m-1]$ which are destroyed. That is to say, $p[1..m-1]$ is k-periodic while $p[1..m]$ is not k-periodic, and furthermore, if $p[1..m-1]$ is l-periodic for some $l < k$, then $p[1..m]$ is also l-periodic.*

▶ **Lemma 24.** *Let $P_i(\lambda)$ be the characteristic polynomial of the pattern $p[1..i]$. Assume that state q_m of the KMP automaton for p has a transition back to state q_{m-k+1} where $1 \le k \le m$. Then*

$$P_m(\lambda) = \begin{cases} \lambda P_{m-1}(\lambda) - (-1)^{wt(p[1..k])} P_{m-k}(\lambda), & \text{if } k < m, \\ \lambda P_{m-1}(\lambda) + (-1)^{wt(p[1..m])}, & \text{if } k = m. \end{cases}$$

5 Conclusions

In this paper, we determined the decision tree complexity of string matching problem for almost every string, except for those string the adversary method fails to give a lower bound, whose fraction is negligible.

The algebraic approach in Section 4 further proves that a few of these strings are evasive. One open problem is to resolve the remaining cases.

The characteristic polynomial of a pattern p, which we encountered in Section 4.3, might be of independent interest itself. We have shown that this polynomial is related to the pattern's periodic behaviour, and it will be interesting to investigate whether other properties of strings can be related to it.

Another natural extension is to consider randomized algorithms. All algorithms proposed in this paper are deterministic, and randomized complexity is still widely open.

References

1 M. Artin. *Algebra.* Pearson Prentice Hall, 2011.

2 J. Berstel and J. Karhumäki. Combinatorics on words - a tutorial. *Bulletin EATCS*, pages 178–228, February 2003.

3 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction To Algorithms.* MIT Press, 2001.

4 Z. Galil and J. Seiferas. Time-space-optimal string matching. *Journal of Computer and System Sciences*, 26(3):280–294, 1983. doi:10.1016/0022-0000(83)90002-8.

5 V. Halava, T. Harju, M. Hirvensalo, and J. Karhumäki. Skolem's problem - on the border between decidability and undecidability. *TUCS Technical Reports 683*, 2005.

6 R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987. doi:10.1147/rd.312.0249.

7 D. Knuth, J. Morris, and V. Pratt. Fast Pattern Matching in Strings. *SIAM Journal on Computing*, 6(2):323–350, jun 1977. doi:10.1137/0206024.

8 P. Nielsen. A note on bifix-free sequences (Corresp.). *IEEE Transactions on Information Theory*, 19(5):704–706, 1973. doi:10.1109/TIT.1973.1055065.

9 R. L. Rivest. On the Worst-Case Behavior of String-Searching Algorithms. *SIAM Journal on Computing*, 6(4):669–674, 1977. doi:10.1137/0206048.

10 R. L. Rivest and J. Vuillemin. A Generalization and Proof of the Aanderaa-Rosenberg Conjecture. In *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*, STOC '75, pages 6–11, New York, NY, USA, 1975. ACM. doi:10.1145/800116.803747.

11 Z. Tuza. Worst-case behavior of string-searching algorithms. *Journal of Statistical Planning and Inference*, 6(1):99–103, 1982. doi:10.1016/0378-3758(82)90060-X.

12 A. Yao. The Complexity of Pattern Matching for a Random String. *SIAM Journal on Computing*, 8(3):368–387, 1979. doi:10.1137/0208029.

Decremental SPQR-trees for Planar Graphs

Jacob Holm[1]
University of Copenhagen, Denmark
jaho@di.ku.dk
https://orcid.org/0000-0001-6997-9251

Giuseppe F. Italiano[2]
University of Rome Tor Vergata, Italy
giuseppe.italiano@uniroma2.it
https://orcid.org/0000-0002-9492-9894

Adam Karczmarz[3]
University of Warsaw, Poland
a.karczmarz@mimuw.edu.pl
https://orcid.org/0000-0002-2693-8713

Jakub Łącki[4]
Google Research, USA
jlacki@google.com
https://orcid.org/0000-0001-9347-0041

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

Abstract

We present a decremental data structure for maintaining the SPQR-tree of a planar graph subject to edge contractions and deletions. The update time, amortized over $\Omega(n)$ operations, is $O(\log^2 n)$. Via SPQR-trees, we give a decremental data structure for maintaining 3-vertex connectivity in planar graphs. It answers queries in $O(1)$ time and processes edge deletions and contractions in $O(\log^2 n)$ amortized time. The previous best supported deletions and insertions in $O(\sqrt{n})$ time.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms, Theory of computation → Graph algorithms analysis, Theory of computation → Data structures design and analysis

Keywords and phrases Graph embeddings, data structures, graph algorithms, planar graphs, SPQR-trees, triconnectivity

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.46

Related Version A full version of the paper is available at [27], `http://arxiv.org/abs/1806.10772`.

[1] Jacob Holm is supported by Mikkel Thorup's Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research under the Sapere Aude research career programme.
[2] Giuseppe F. Italiano is partially supported by the Italian Ministry of Education, University and Research under Project AMANDA (Algorithmics for MAssive and Networked DAta).
[3] Adam Karczmarz is supported by the grants 2014/13/B/ST6/01811 and 2017/24/T/ST6/00036 of the Polish National Science Center.
[4] When working on this paper Jakub Łącki was partly supported by the EU FET project MULTIPLEX no. 317532 and the Google Focused Award on "Algorithms for Large-scale Data Analysis" and Polish National Science Center grant number 2014/13/B/ST6/01811. Part of this work was done while Jakub Łącki was visiting the Simons Institute for the Theory of Computing.

© Jacob Holm, Giuseppe F. Italiano, Adam Karczmarz, Jakub Łącki, and Eva Rotenberg;
licensed under Creative Commons License CC-BY

1 Introduction

A graph algorithm is called *dynamic* if it is able to answer queries about a given property while the graph is undergoing a sequence of updates, such as edge insertions and deletions. It is *incremental* if it handles only insertions, *decremental* if it handles only deletions, and *fully dynamic* if it handles both insertions and deletions. In designing dynamic graph algorithms, one is typically interested in achieving fast query times (either constant or polylogarithmic), while minimizing the update times. The ultimate goal is to perform fast both queries and updates, i.e., to have both query and update times either constant or polylogarithmic. So far, the quest for obtaining polylogarithmic time algorithms has been successful only in few cases. Indeed, efficient dynamic algorithms with *polylogarithmic* time per update are known only for few problems, such as dynamic connectivity, 2-connectivity, minimum spanning tree and maximal matchings in undirected graphs (see, e.g., [6, 24, 25, 29, 37, 52, 54, 56]). On the other hand, some dynamic problems appear to be inherently harder. For example, the fastest known algorithms for basic dynamic problems, such as reachability, transitive closure, and dynamic shortest paths, have updates that run in only *polynomial* time (see, e.g., [9, 10, 11, 39, 49, 51, 55]).

A similar situation holds for planar graphs where dynamic problems have been studied extensively, see e.g. [3, 14, 16, 18, 20, 21, 26, 34, 42, 43, 44, 45, 53]. Despite this long-time effort, the best algorithms known for some basic problems on planar graphs, such as dynamic shortest paths and dynamic planarity testing, still have polynomial update time bounds. For instance, for fully dynamic shortest paths on planar graphs the best known bound per operation is $\widetilde{O}(n^{2/3})$ amortized [19, 34, 36, 40] (using $\widetilde{O}$-notation to hide polylogarithmic factors), while for fully dynamic planarity testing the best known bound per operation is $O(\sqrt{n})$ amortized [16].

In the last years, this exponential gap between polynomial and polylogarithmic bounds has sparkled some new exciting research. On one hand, it was shown that there are dynamic graph problems, including fully dynamic shortest paths, fully dynamic single-source reachability and fully dynamic strong connectivity, for which it may be difficult to achieve subpolynomial update bounds. This started with the pioneering work by Abboud and Vassilevska-Williams [2], who proved conditional lower bounds based on popular conjectures. Very recently, Abboud and Dahlgaard [1] proved polynomial lower bounds for the update time for dynamic shortest paths also on planar graphs, again based on popular conjectures.

On the other hand, the question of improving the update bounds from polynomial to polylogarithmic, has, for several other dynamic graph problems, received much attention in the last years. For instance, there was a very recent improvement from polynomial to polylogarithmic bounds for decremental single-source reachability (and strongly connected components) on planar graphs: more precisely, the improvement was from $O(\sqrt{n})$ amortized [42] to $O(\log^2 n \log\log n)$ amortized [33] (both amortizations are over sequences of $\Omega(n)$ updates). Other problems that received a lot of attention are fully dynamic connectivity and minimum spanning tree in general graphs. Up to very recently, the best worst-case bound for both problems was $O(\sqrt{n})$ per update [15]: since then, much effort has been devoted towards improving this bound (see e.g., [37, 38, 47, 48, 57]).

In this paper, we follow the ambitious goal of achieving polylogarithmic update bounds for dynamic graph problems. In particular, we show how to improve the update times from polynomial to polylogarithmic for another important problem on planar graphs: decremental 3-vertex connectivity. Given a graph $G = (V, E)$ and two vertices $x, y \in V$ we say that x and

y are 2-vertex connected (or, as we say in the following, *biconnected*) if there are at least two vertex-disjoint paths between x and y in G. We say that x and y are 3-vertex connected (or, as we say in the following, *triconnected*) if there are at least three vertex-disjoint paths between x and y in G. The decremental planar triconnectivity problem consists of maintaining a planar graph G subject to an arbitrary sequence of edge deletions, edge contractions, and query operations which test whether two arbitrary input vertices are triconnected. We remark that decremental triconnectivity on planar graphs is of particular importance. Apart from being a fundamental graph property, a triconnected planar graph has only one planar embedding, a property which is heavily used in graph drawing, planarity testing and testing for isomorphism [31, 32, 35]. Furthermore, our extended repertoire of operations, which includes edge contractions, contains all operations needed to obtain a graph minor, which is another important notion for planar graphs.

While polylogarithmic update bounds for decremental 2-edge and 3-edge connectivity, and for decremental biconnectivity on planar graphs have been known for more than two decades [20], decremental triconnectivity on planar graphs presents some special challenges. Indeed, while connectivity cuts for 2-edge and 3-edge connectivity, and for biconnectivity have simple counterparts in the dual graph or in the vertex-face graph (see Section 2 for a formal definition of vertex-face graph), triconnectivity cuts (separation pairs, i.e., pairs of vertices whose removal disconnects the graph) have a much more complicated structure in planar graphs. Roughly speaking, maintaining 2-edge and 3-edge connectivity cuts in a planar graph under edge deletions corresponds to maintaining respectively self-loops and cycles of length 2 (pairs of parallel edges) in the dual graph under edge contractions. Similarly, maintaining biconnectivity and triconnectivity cuts in a planar graph under edge deletions corresponds to maintaining, respectively, cycles of length 2 and cycles of length 4 in the vertex-face graph. While detecting cycles of length 2 boils down to finding duplicates in the multiset of all edges, detecting cycles of length 4 under edge contractions is far more complex. We believe that this is the reason why designing a fast solution for decremental triconnectivity on planar graphs has been an elusive goal, and the best bound known of $O(\sqrt{n}\,)$ per update [17] has been standing for over two decades.

Our results and techniques. Our main result is given in the following theorem.

▶ **Theorem 1.** *There is a data structure that can be initialized on a planar graph G on n vertices and $O(n)$ edges in $O(n \log n)$ time, and support any sequence of $\Omega(n)$ edge deletions or contractions in total time $O(n \log^2 n)$, while supporting queries to pairwise triconnectivity in worst-case constant time per query.*

This is an exponential speed-up over the previous $O(\sqrt{n}\,)$ long-standing bound [17]. To obtain our bounds, we also need to solve decremental biconnectivity on planar graphs in constant time per query and $O(\log^2 n)$ amortized time per edge deletion or contraction. (A better $O(\log n)$ amortized bound can be obtained if no contractions are allowed [26].) In the description we assume that the graph is embedded in the plane (a so-called *plane* graph). However, the data structure may handle an arbitrary planar (non-embedded) graph by first embedding the initial graph in the plane in linear time. This choice of initial embedding has no effect on either the queries, or on which edge deletions or contractions are possible.

Our results are obtained using two new tools, which may be of independent interest. The first tool is an algorithm for efficiently detecting and reporting cycles of length 4 as they arise in a dynamic plane graph subject to edge contractions and insertions. The algorithm works for a graph with bounded face-degree, i.e, where each face is delimited by at most some constant number of edges. Specifically, given a plane graph with bounded face-degree subject to edge-contractions and edge-insertions across a face, we can maintain the set of edges lying on cycles of length at most 4. The total running time is $O(n \log n)$. One of the challenges that we face is that a plane graph may have as many as $\Omega(n^2)$ distinct cycles of length 4. Still, we give a surprisingly simple algorithm for solving this problem. The difficulty of the algorithm lies in the analysis — in fact, this analysis is the most technically involved part of this paper.

The second tool is a new data structure that maintains the SPQR-tree [12] of each biconnected component of a planar graph subject to edge deletions and edge contractions, in $O(\log^2 n)$ amortized time per operation. While incremental algorithms for maintaining the SPQR-tree were known for more than two decades [12, 13], to the best of our knowledge no decremental algorithm was previously known.

Organization of the paper. The remainder of the paper is organized as follows. In Section 2, we introduce notation and definitions that we later use. Then, in Section 3 we present a high-level overview of our results. Finally, in Section 4 we give more details of our algorithm for maintaining an SPQR-tree under edge deletions and contractions.

Due to space constraints, the algorithm for detecting cycles of length 4 under contractions, which is a key tool in maintaining an SPQR-tree, is deferred to the full version [27], along with the detailed discussion of how to use the SPQR-trees to maintain information about triconnectivity, and a selections of proofs omitted from Section 4.

2 Preliminaries

Throughout the paper we use the term *graph* to denote an undirected *multigraph*, that is, we allow the graphs to have parallel edges and self-loops. Formally, each edge e of such a graph is a pair $(\{u, w\}, \mathrm{id}(e))$ consisting of a pair of vertices and a unique *integer identifier* used to distinguish between the parallel edges. For simplicity, in the following we skip the identifier and use just uw to denote one of the edges connecting vertices u and w. If the graph contains no parallel edges and no self-loops, we call it *simple*.

Given a graph G, we use $V(G)$ to denote the vertices, and $E(G)$ to denote the edges of G. For $e \in E(G)$, we use $G - e$ to denote the graph obtained from G by removing e. If e is not a self-loop, we use G/e to denote the graph obtained by contracting e. A *cycle* C of length $|C| = k$ in a graph G is a cyclic sequence of edges $C = e_1, e_2 \ldots, e_k$ where $e_i = u_i u_{i+1}$ for $1 \le i < k$ and $e_k = u_k u_1$. Note that this definition allows cycles of length 1 (a self-loop) or 2 (a pair of parallel edges). A cycle is *simple* if $\mathrm{id}(e_i) \neq \mathrm{id}(e_j)$ and $u_i \neq u_j$ for $i \neq j$. We sometimes abuse notation and treat a cycle as a set of edges or a cyclic sequence of vertices.

The *components* of a graph G are the minimal subgraphs $H \subseteq G$ such that for every edge $uv \in E(G)$, $u \in V(H)$ if and only if $v \in V(H)$. The components of a graph partition the vertices and edges of the graph. A graph G is *connected* if it consists of a single component. For a positive integer k, a graph is *k-vertex connected* if and only if it is connected, has at least k vertices, and stays connected after removing any set of at most $k - 1$ vertices. The *local vertex connectivity* of a pair of vertices u, v, denoted $\kappa(u, v)$, is the maximal number of internally vertex-disjoint u, v-paths. By Menger's Theorem [46], G is k-vertex connected if and only if $\kappa(u, v) \ge k$ for every pair of non-adjacent vertices u, v. We say that u, v are (locally) k-vertex connected if $\kappa(u, v) \ge k$. We follow the common practice of using *biconnected* as a synonym for 2-vertex connected and *triconnected* as a synonym for 3-vertex connected. An *articulation point* v of G is a vertex whose removal increases the number of

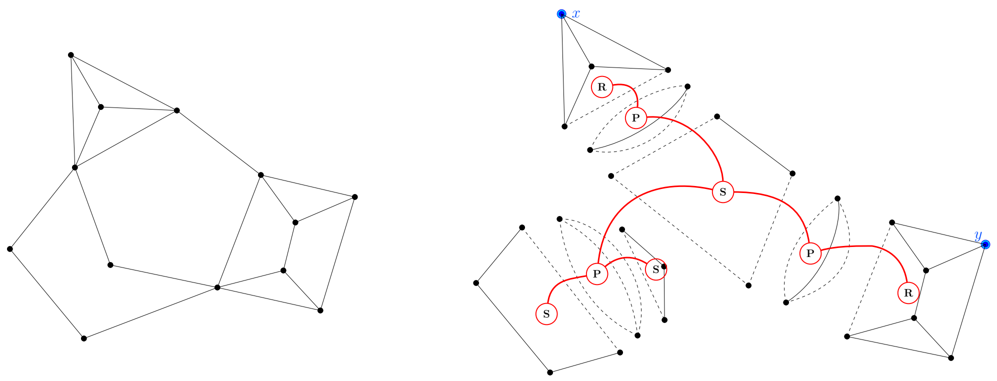


Figure 1 A biconnected graph and its SPQR-tree. Note that adding the edge xy would collapse a path of SPQR-nodes into one. Deletion can thus result in the opposite transformation.

components of G. Thus, a graph is biconnected if and only if it is connected and has no articulation points.

The structure of the biconnected components of a connected graph can be described by a tree called the *block-cutpoint tree* [23, p. 36], or *BC-tree* for short. This tree has a vertex for each biconnected component (block) and for each articulation point of the graph, and an edge for each pair of a block and an articulation point that belongs to that block.

We recall that a graph G that is biconnected but not triconnected has at least one separation pair, i.e., a pair of vertices that can be removed to disconnect G:

▶ **Definition 2** (Hopcroft and Tarjan [30, p. 6])**.** Let $\{a, b\}$ be a pair of vertices in a biconnected multigraph G. Suppose the edges of G are divided into equivalence classes $E_1, E_2, \dots, E_k$, such that two edges which lie on a common path not containing any vertex of $\{a, b\}$ except as an end-point are in the same class. The classes E_i are called the *separation classes* of G with respect to $\{a, b\}$. If there are at least two separation classes, then $\{a, b\}$ is a *separation pair* of G unless (i) there are exactly two separation classes, and one class consists of a single edge, or (ii) there are exactly three classes, each consisting of a single edge[5].

The notion of the block cutpoint tree over biconnected components can be generalised to an *SPQR*-tree over triconnected components as follows:

▶ **Definition 3.** The SPQR-tree for a biconnected multigraph $G = (V, E)$ with at least 3 edges is a tree with nodes labeled S, P, or R, where each node x has an associated *skeleton graph* $\Gamma(x)$ with the following properties:

- For every node x in the SPQR-tree, $V(\Gamma(x)) \subseteq V$.
- For every edge $e \in E$ there is a unique node x in the SPQR-tree such that $e \in E(\Gamma(x))$.
- For every edge (x, y) in the SPQR-tree, $V(\Gamma(x)) \cap V(\Gamma(y))$ is a separation pair $\{a, b\}$ in G, and there is a *virtual edge* ab in each of $\Gamma(x)$ and $\Gamma(y)$ that corresponds to (x, y).
- For every node x in the SPQR-tree, every edge in $\Gamma(x)$ is either in E or a virtual edge.
- If x is an S-node, $\Gamma(x)$ is a simple cycle with at least 3 edges.
- If x is a P-node, $\Gamma(x)$ consists of a pair of vertices with at least 3 parallel edges.
- If x is an R-node, $\Gamma(x)$ is a simple triconnected graph.
- No two S-nodes are neighbors, and no two P-nodes are neighbors.

[5] These two exceptions actually make it easier to state some properties related to separation pairs.

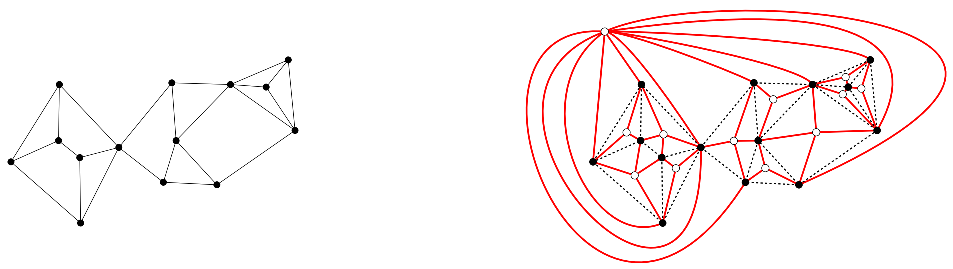


Figure 2 Left: a plane graph. Right: the corresponding vertex-face graph (red) and the underlying graph (dashed).

The SPQR-tree for a biconnected graph is unique (see e.g. [12]). The (skeleton graphs associated with) the SPQR-nodes are sometimes referred to as G's triconnected components.

Let G be a plane graph (a planar graph embedded in the plane). For each component H of G, let H^* denote the dual graph of H, defined as the graph obtained by creating a vertex for each face in the embedding of H, and an edge e^* (called the *dual edge* of e), connecting the two (not necessarily distinct) faces that e is incident to. Let G^* denote the graph obtained from G by taking the dual of each component.

Each face f in a plane graph is bounded by a (not necessarily simple) cycle called the *face cycle* for f. We call the length of this cycle the *face-degree* of f. We call any other cycle a *separating cycle*.

Let G be a connected plane multigraph with at least one edge. Define the set $E^\diamond(G)$ of *corners*[6] of G to be the the set of ordered pairs of (not necessarily distinct) edges (e_1, e_2) such that e_1 immediately precedes e_2 in the clockwise order around some vertex, denoted $v(e_1, e_2)$. Note that if $(e_1, e_2) \in E^\diamond(G)$, then $(e_2^*, e_1^*) \in E^\diamond(G^*)$. We denote by $G^\diamond$ the *vertex-face* graph[7] of G (see Figure 2). This is a plane multigraph with vertex set $V(G) \cup V(G^*)$, and an edge between $v(e_1, e_2)$ and $v(e_2^*, e_1^*)$ for each corner $(e_1, e_2) \in E^\diamond(G)$. Abusing notation slightly, we can write $G^\diamond$ as $= (V(G) \cup V(G^*), E^\diamond(G))$. We use the following well-known facts about the vertex-face graph:

1. $G^\diamond$ is bipartite and plane, with a natural embedding given by the embedding of G.
2. The vertex-face graphs of G and G^* are the same: $G^\diamond = (G^*)^\diamond$.
3. There is a one-to-one correspondence between the edges of G and the faces of $G^\diamond$ (in the natural embedding, each face of $G^\diamond$ contains exactly one edge of G interior, see Fig 2).
4. $(G^\diamond)^*$ (also known as the *medial graph*) is 4-regular.
5. $G^\diamond$ is simple if and only if G is loopless and biconnected (See e.g. [8, Theorem 5(i)]).
6. $G^\diamond$ is simple, triconnected and has no separating 4-cycles if and only if G is simple and triconnected (See e.g. [8, Theorem 5(iv)]).

If v is an articulation point in G or has a self-loop, then in any planar embedding of G there is at least one face f whose face cycle contains v at least twice. Any such f is either an articulation point or has a self-loop in G^*, and v and f are connected by (at least) two edges in $G^\diamond$.

The dynamic operations on G correspond to dynamic operations on G^* and $G^\diamond$. Deleting a non-bridge edge e of G corresponds to contracting e^* in G^*, that is, $(G-e)^* = G^*/e^*$. Similarly, contracting an edge e corresponds to deleting e^* from the dual, so $(G/e)^* = G^* - e^*$.

[6] For alternative definitions, see e.g. [28] and [50]. The latter uses *angles* for what we call corners.
[7] A.k.a. the *vertex-face incidence graph* [7], the *angle graph* [50], and the *radial graph* [5].

Finally, deleting a non-bridge edge or contracting an edge corresponds to adding and then immediately contracting an edge across a face of $G^\diamond$ (and removing two duplicate edges).

Finally, the useful concept of a separation is well-defined, even for general graphs:

▶ **Definition 4.** Given a graph $G = (V, E)$, a *separation* of G is a pair of vertex sets (V', V'') such that the induced subgraphs $G' = G[V'], G'' = G[V'']$ contain all edges of G, and $V' \setminus V''$ and $V'' \setminus V'$ are both nonempty. A separation is *balanced* if $\max \{|V'|, |V''|\} \leq \alpha |V|$ for some fixed constant $\frac{1}{2} \leq \alpha < 1$. If (V', V'') is a separation of G, the set $S = V' \cap V''$ is called a *separator* of G. A separator S is *small* if $|S| = O(\sqrt{|V|})$, and it is a *cycle separator* if the subgraph of G induced by S is Hamiltonian.

Note that a *separation*, which is a pair of vertex sets, should not be confused with a *separation pair*, which is a pair of vertices (see Definition 2).

3 Overview of Our Approach

The SPQR-tree naturally reflects the triconnected components of the graph, so it is perhaps not surprising that an SPQR-tree can be augmented to answer pairwise triconnectivity queries in constant time. The challenge is to update the SPQR-tree under decremental updates. For this, we need a way to find all new separation pairs that arise. These separation pairs are related to separating 4-cycles in the vertex-face graph, in which decremental updates correspond to "collapsing" faces, i.e. the addition and immediate contraction of an edge across a face. So, the core of our approach is an algorithm for detecting separating 4-cycles in a particular kind of plane graph subject to valid edge insertions and contractions.

Detecting separating 4-cycles. A 4-cycle is a simple cycle of length 4. We say that a 4-cycle in a plane graph G is a *face* 4*-cycle* if it is a cycle bounding a face of G, and a *separating* 4*-cycle* otherwise. There is a one-to-one correspondence between separation pairs in G and separating 4-cycles in the vertex-face graph $G^\diamond$. (See [27] for details.)

Since no two parallel edges can lie on the same 4-cycle, and no self-loop can be contained in a 4-cycle, we can assume the input graph is simple. However, when we contract edges, new parallel edges and self-loops may arise. To handle this, we could detect and remove all parallel edges, but it turns out that both the algorithm and the analysis become simpler if we keep (most of) the additional edges, as long as no two parallel edges are consecutive in the circular ordering around both their endpoints. This is captured by the following definition.

▶ **Definition 5.** A plane graph is *quasi-simple* if the dual of each non-simple component has minimum degree 3. (In [41] these graphs are called *semi-strict*.)

Roughly speaking, a quasi-simple graph is obtained from a plane multigraph by merging parallel edges that lie next to each other in the circular orderings around both their endpoints.

We build a structure for 4-cycle detection by recursively using balanced separators, and by detecting, for each separator, the cycles that cross the separator. Detecting 4-cycles that cross a separator is not trivial, and our analysis introduces a complicated potential function which reflects how well connected the non-separator vertices are with the separator, that is, how many neighbors on the separator they have. At the same time, we make sure that all the work done can be paid with the decrease in the potential. Our analysis exploits the fact that for a subset of vertices S in quasi-simple planar graph, at most $O(|S|)$ vertices have 4 or more neighbors in S. Specifically, this holds when S is the set of separator vertices.

The recursive use of separators can be sketched as follows: Let S be a small balanced separator in $G = (V, E)$ that induces a separation (V_1, V_2), that is, $V_1 \cap V_2 = S$ and

$V_1 \cup V_2 = V$. Moreover, let $n = |V|$. We observe that each 4-cycle is fully contained in V_1 or V_2, or consists of two paths of length 2 that connect vertices of S. This motivates the following recursive approach. We compute a separator S of $O(\sqrt{n})$ vertices and then find all paths of length 2 that connect vertices of S. Since the size of S is $O(\sqrt{n})$, there are only $O(n)$ pairs of vertices of S, and for each pair of vertices, we can easily check if the two-edge paths connecting them form any separating 4-cycles. It then remains to find the 4-cycles that are fully contained in either V_1 or V_2, which can be done recursively. Because S is a balanced separator, the recursion has $O(\log n)$ levels.

This algorithm can be made dynamic under contractions and edge insertions that respect the embedding of G. Contractions are easy to handle, as they preserve planarity. Moreover, a separator S of a planar graph can be easily updated under contractions. Namely, whenever an edge uw is contracted, the resulting vertex belongs to the separator iff any of u and w did. Insertions that preserve planarity, however, are in general harder to accommodate. To handle this we introduce a new type of separators that we call *face-preserving* separators, which (like cycle-separators) always exist when the face-degree is bounded. These are still preserved by contractions, but also ensure that any edge across a face can be inserted.

All in all, there are $O(\log n)$ levels of size $O(n)$ each, where each level handles insertions and contractions in constant time, leading to a total of $O(n \log n)$ time. (See [27] for details.)

▶ **Theorem 6.** *Let G be an n-vertex connected quasi-simple plane graph with bounded face degree. There exists a data structure that maintains G under contractions and embedding-respecting insertions, and after each update operation reports edges that become members of some separating 4-cycle. It runs in $O(n \log n)$ total time.*

Maintaining SPQR-trees. The main challenge in maintaining an SPQR-tree is handling the case when an edge within a triconnected component is deleted. First of all, the data structure should be able to detect whether or not the component is still triconnected.

For the skeleton Γ of any R-node in the SPQR-tree of G, we maintain a 4-cycle detection structure for the corresponding vertex-face graph $\Gamma^\diamond$. A separating 4-cycle in $\Gamma^\diamond$ corresponds to a separation pair in Γ, which would witness that Γ is no longer triconnected. The deletion or contraction of the edge e in the triconnected component Γ of G corresponds to collapsing a face in $\Gamma^\diamond$ by the insertion and immediate contraction of an edge. By detecting new 4-cycles in $\Gamma^\diamond$, we can therefor detect when the corresponding triconnected component falls apart.

However, this is not the only challenge. If Γ does indeed cease to be triconnected, the SPQR-tree of $(\Gamma - e)$ (or (Γ/e) when doing a contraction) is a path H. This is where we need the 4-cycle detection structure to output the edges contained in separating 4-cycles. Those edges correspond to a set of corners N of G. We use those corners to guide a search, which identifies the non-largest components of the SPQR-path H. More specifically, if a vertex v now belongs to two distinct triconnected components, there are two corners in N that separate the edges incident to v into two groups of edges, each belonging to a distinct triconnected component. We can afford to build a 4-cycle detection structure for $\Gamma'^\diamond$ for any non-largest triconnected component Γ' on the path from scratch. To obtain the data structure representing the largest component, we delete or contract the edges of the smaller components from Γ, while updating $\Gamma^\diamond$. Since an edge only becomes part of a structure built from scratch when its triconnected component size has been halved, this happens only $O(\log n)$ times per edge. Since the time spent on building 4-cycle detection structures is $O(\log n)$ per contributing edge, the total time becomes $O(n \log^2 n)$.

Finally, since no two S-nodes can be neighbors and no two P-nodes can be neighbors, some S- or P-nodes in H may have to be merged with their (at most 2) neighbors of the same type outside H. To handle this step efficiently, we keep the SPQR-tree rooted in an arbitrary node. While merging the skeleton graphs of two S- or P-nodes can be done in constant time, it is more costly to update the parent pointers in the children of the merged nodes. Hence, we move the children of the node with fewer children to the other node. This way, each node changes parent at most $O(\log n)$ times before it is deleted or split. The total number of distinct SPQR-nodes that exist throughout the lifetime of the data structure is $O(n)$, so the total time used for maintaining the parent pointers is $O(n \log n)$.

Since SPQR-trees are only defined for biconnected graphs, another challenge is to maintain SPQR-trees for each biconnected component, even as the decremental update operations cause the biconnected components to fall apart. We thus maintain also the BC-tree of the graph (see Section 2). If the BC-tree is rooted arbitrarily at any block, each non-root block has a unique articulation point separating it from its parent.

To handle updates, we notice that the SPQR-tree points to the fragile places where the graph is about to cease to be biconnected: An edge deletion in an S-node will break up a block in the BC-tree into a path, and an edge contraction in a P-node breaks a block in the BC-tree into a star. Upon such an update, we remove the aforementioned S- or P-node from the SPQR-tree, breaking it up into an SPQR-forest. Each tree corresponds to a new block in the BC-tree. They form a path (or a star), and the ordering along the path, as well as the articulation points, can be read directly from the SPQR-tree. (See Section 4 for details.)

On the other hand, in order to even know which SPQR-tree to modify during an update, we can search in the BC-tree for the right SPQR-structure in which to perform the operation.

Bi- and triconnectivity. Finally, we use SPQR-trees to facilitate triconnectivity queries. First of all, vertices need to be biconnected in order to be triconnected. In the rooted BC-tree, assign each vertex to its root-nearest block. It is enough that each vertex knows the name of its block, and each block knows the vertex separating it from its parent. Then, any two vertices are biconnected if and only if they either have the same block, or one is the unique vertex separating the block of the other from its parent.

For triconnectivity, the maintained information, as well as the query handling, is similar, using the SPQR-tree in place of the BC-tree. Namely: each non-root node in the SPQR-tree stores the *virtual edge* (see Definition 3) that separates it from its parent. Each vertex knows the root-nearest node containing it, and, if this is an S-node, its at most two children containing the vertex.

The main challenge is to handle updates. Note that the change to the SPQR-tree may involve both the split and merge of nodes. In particular, we have one split and up to several merges when a triconnected component falls apart into an SPQR-path. However, upon a merge, we can afford to update the information regarding vertices in the non-largest components, costing only an additive $\log n$ to the amortized running time. Similarly, upon a split, we update any information that relates to vertices in the non-largest components only.

The total running time is thus $O(n \log n + f(n))$, where $f(n)$ is the running time for maintaining the SPQR-tree. (See [27] for details.)

▶ **Theorem 1.** *There is a data structure that can be initialized on a planar graph G on n vertices and $O(n)$ edges in $O(n \log n)$ time, and support any sequence of $\Omega(n)$ edge deletions or contractions in total time $O(n \log^2 n)$, while supporting queries to pairwise triconnectivity in worst-case constant time per query.*

Algorithm 1 Removing an edge e from a P-node x of T.

```
function REMOVEP($e, x, T$)
    remove $e$ from $\Gamma(x)$
    if $\Gamma(x)$ has two edges then
        if $\Gamma(x)$ has no virtual edges then
            delete $T$
        else if $\Gamma(x)$ has one virtual edge then
            $y :=$ the only neighbor of $x$
            $e_x :=$ the virtual edge in $\Gamma(y)$ corresponding to $x$
            replace $e_x$ by the non-virtual edge of $\Gamma(x)$
            remove $x$ from $T$
        else if $\Gamma(x)$ has two virtual edges then
            $\{y, z\} :=$ neighbors of $x$ in $T$
            remove $x$ from $T$, making $y$ and $z$ neighbors in $T$
            if $y$ and $z$ are $S$-nodes then
                merge $y$ and $z$ into one node
```

4 Decremental SPQR-trees

In this section, we use the data structure of Theorem 6 to maintain an SPQR-tree (see Definition 3) for each biconnected component of G with at least 3 edges under arbitrary edge deletions and contractions. We start with some useful facts.

▶ **Lemma 7.** *Let G be a biconnected graph. If a 4-cycle $C = (v_1, f_1, v_2, f_2)$ in $G^\diamond$ is a separating cycle, then v_1, v_2 is a separation pair of G and f_1, f_2 is a separation pair of G^*.*

▶ **Lemma 8.** *Let G be a loopless biconnected plane graph and u, w be a separation pair in G. Consider the set of edges E_x incident to $x \in \{u, w\}$. Then, the edges of E_x belonging to each separation class of u, w are consecutive in the circular ordering around both u and w.*

▶ **Lemma 9.** *Let G be a triconnected plane graph and $e = uw \in E(G)$. Assume that $G - e$ is not triconnected. Then, the SPQR-tree of $G - e$ is a path H (we call it an SPQR-path). Moreover, given all edges that lie on 4-cycles in $(G - e)^\diamond$, we can compute all nodes of H (i.e., their skeleton graphs) except for the largest one in time that is linear in their size.*

For a planar graph, there is a nice duality, as proven by Angelini et al. [4, Lemma 1]. Define the dual SPQR-tree as the tree obtained from the SPQR-tree by interchanging S- and P-nodes, and taking the dual of the skeletons.

▶ **Lemma 10** (Angelini et al [4]). *The SPQR-tree of G^* is the dual SPQR-tree of G.*

Let G be a connected plane graph. Since $(G^\diamond)^*$ is 4-regular, $G^\diamond$ is quasi-simple and has bounded face-degree. Furthermore, any edge deletion or contraction in G that leaves G connected, corresponds to an edge insertion and immediate contraction in $G^\diamond$. Thus by Theorem 6 we can maintain a data structure for G under connectivity-preserving edge deletions and contractions, that after each update operation reports the corners that become part of a separating 4-cycle in $G^\diamond$.

In the algorithm we maintain one SPQR-tree for each biconnected component with at least 3 edges. We now describe how these trees are updated upon edge deletions. The procedures, depending on the type of the SPQR-tree node are given as Algorithms 1, 2 and 3. Note that the lines 4 and 5 in Algorithm 2 only introduce notation, that is the values of the variables are not computed. (See [27] for a proof of correctness.)

Algorithm 2 Removing an edge e from an R-node x of T.

```
1: function REMOVER($e, x, T$)
2:     remove $e$ from $\Gamma(x)$
3:     if $\Gamma(x)$ has a separation pair then
4:         $X'$ := SPQR-path representing $\Gamma(x)$
5:         $x_{big}$ := the node of $X'$ st. $\Gamma(x_{big})$ has the most edges
6:         compute all nodes of $X' \setminus x_{big}$
7:         remove and contract edges of $\Gamma(x)$ to obtain $\Gamma(x_{big})$
8:         replace $x$ in $T$ by $X'$ (connect each child of $x$ to the correct node of $X'$)
9:         for each $S$- or $P$-node $z \in X'$ do
10:            for each neighbor $z' \notin X'$ do
11:                if $z, z'$ are same type then
12:                    merge $z$ with $z'$
```

Algorithm 3 Removing an edge e from an S-node x of T.

```
1: function REMOVES($e, x, T$)
2:     remove $e$ from $\Gamma(x)$
3:     remove $x$ from $T$
4:     for each edge $e'$ in $\Gamma(x)$ do
5:         Make a new BC-node $z$
6:         if $e'$ is a virtual edge then
7:             $y$ := neighbor of $x$ in $T$ corresponding to $e'$
8:             Make the tree containing $y$ the SPQR-tree for the new BC-node
9:             if $y$ is a $P$-node then
10:                removeP($y, e', T$)
11:            else
12:                removeR($y, e', T$)
```

We can now prove the main theorem of this section. Note that, as in the block-cutpoint tree, we root each SPQR-tree in an arbitrary vertex.

▶ **Theorem 11.** *There is a data structure that can be initialized on a simple planar graph G on n vertices in $O(n \log n)$ time, and supports any sequence of edge deletions or contractions in total time $O(n \log^2 n)$, while maintaining an explicit representation of a rooted SPQR-tree for each biconnected component with at least 3 edges, including all the skeleton graphs for the triconnected components. Moreover, during updates, the total number of times a node of an SPQR-tree changes its parent is $O(n \log n)$.*

Proof. We first partition the graph into biconnected components, and, as sketched in Section 3, maintain the block-cutpoint tree explicitly. Thus, given two vertices u, v, we can in $O(1)$ time access the biconnected component containing both of them, along with its auxiliary data. Now, for each biconnected component C_i, we compute the SPQR-tree T. This can be done in linear time due to [22]. We also root each SPQR-tree in an arbitrary node, and keep the trees rooted as they are updated.

For each node x of T we maintain the graph $\Gamma(x)$. Each virtual edge of $\Gamma(x)$ has a pointer to the neighbor of x it represents. Moreover, for each R-node r, we keep a data structure of Theorem 6 for detecting separating 4-cycles in the vertex-face graph $(\Gamma(r))^\diamond$. By Lemma 7, any separating 4-cycle in $(\Gamma(r))^\diamond$ corresponds to a separation pair in $\Gamma(r)$. Since r is an R-node, there are no separating 4-cycles to begin with, but some may appear after an update.

Since the total size of the R-components is n, it follows from Theorem 6 that the entire construction time is $O(n \log n)$.

Deletion. When an edge e is removed we find the node x of the SPQR-tree, such that e is a non-virtual edge in x. Then, we proceed according to Algorithms 1, 2 and 3.

Whenever an edge fg is deleted from an R-node r, we update the corresponding 4-cycle detection structure for $(\Gamma(r))^\diamond$. We first insert the dual edge $(fg)^*$ in the vertex-face graph, and then contract along that edge. This allows us to detect whether $\Gamma(r)$ has any separation pairs after each edge deletion.

Let us now analyze the running time. When processing an edge deletion, the following changes can take place in a SPQR-tree (all other changes can be handled in $O(1)$ time):

- an R-node is split into multiple nodes,
- two P-nodes or S-nodes are merged,
- an S- or P- node is deleted.

Note, a P- or S-node can never get split. So, though each edge may at first belong to nodes that are split, once it becomes a part of a P- or S-node, its node only participates in merges.

When two S- or P-nodes are merged, we can merge their skeleton graphs in constant time. These skeleton graphs have only two common nodes, and their lists of adjacent edges can be merged in constant time thanks to Lemma 8. When nodes are merged, we also have to update the parent pointers of their children. To bound the number of these updates, we merge the node with fewer children into the node with more. Thus, the number of parent updates caused by these merges is $O(n \log n)$, and so is the impact on the running time.

A similar analysis applies to the case when an R-node r is split into an SPQR-path. By Lemma 9, we can compute all but the largest node of the SPQR-path in linear time. Since the size of the skeleton graph in each of these nodes is at most half the size of $\Gamma(r)$, each edge takes part in this computation at most $O(\log n)$ times. For every new R-nodes, we also initialize their associated data structures for detecting 4-cycles. We charge the running time of each data structure to this initialization. From Theorem 6 we get that recomputing all the nodes and data structures takes $O(n \log^2 n)$ total time.

Taking care of the largest component of the SPQR-path is even easier, as we can simply reuse the skeleton graph of r and its associated data structure for detecting 4-cycles. To update the skeleton graph, we use the following lemma.

▶ **Lemma 12.** *If G is triconnected, $e \in E(G)$, and x is an R-node in the SPQR-tree for $G - e$, then there exists a sequence of $|E(G)| - |E(\Gamma(x))|$ edge deletions and contractions that transform $G - e$ into $\Gamma(x)$ while keeping the graph connected at all times.*

After an R-node r is split into a SPQR-path H we also need to update the parent pointers in the children of r. However, the number of children to update is at most the number of edges in the non-largest components of the SPQR-path. As we have argued, the total number of such edges across all deletions is $O(n \log n)$.

Contraction. The contraction of an edge of the plane graph G corresponds to the deletion of an edge of its dual graph, G^*. By Lemma 10, the SPQR-tree of G^* is the dual SPQR-tree of G. Thus, if the edge was in a P-node of the SPQR-tree, its contraction is handled like the deletion of an edge in a S-node, and vice versa.

If the contracted edge e belongs to an R-node, that R node may expand to a path in the SPQR-tree (because deletion in G^* may expand an R-node into a path). In the vertex-face graph, we may find all edges participating in new separating 4-cycles, corresponding to separating corners of the graph. To find the new components, we simply apply Lemma 9 to the dual graph and proceed analogously to a deletion. ◀

References

1 Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 477–486, 2016. doi:10.1109/FOCS.2016.58.

2 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014. doi:10.1109/FOCS.2014.53.

3 Ittai Abraham, Shiri Chechik, and Cyril Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1199–1218, 2012. doi:10.1145/2213977.2214084.

4 Patrizio Angelini, Thomas Bläsius, and Ignaz Rutter. Testing mutual duality of planar graphs. *Int. J. Comput. Geometry Appl.*, 24(4):325–346, 2014. doi:10.1142/S0218195914600103.

5 Dan Archdeacon and R Bruce Richter. The construction and classification of self-dual spherical polyhedra. *J. Comb. Theory, Series B*, 54(1):37–63, 1992. doi:10.1016/0095-8956(92)90065-6.

6 Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in $O(\log n)$ update time. *SIAM J. Comput.*, 44(1):88–113, 2015. doi:10.1137/130914140.

7 Graham R. Brightwell and Edward R. Scheinerman. Representations of planar graphs. *SIAM J. Discrete Math.*, 6(2):214–229, 1993. doi:10.1137/0406017.

8 Gunnar Brinkmann, Sam Greenberg, Catherine Greenhill, Brendan D. Mckay, Robin Thomas, and Paul Wollan. Generation of simple quadrangulations of the sphere. *Discrete Math.*, 305(1-3):33–54, 2005. doi:10.1016/j.disc.2005.10.005.

9 Shiri Chechik, Thomas Dueholm Hansen, Giuseppe F. Italiano, Jakub Łącki, and Nikos Parotsidis. Decremental single-source reachability and strongly connected components in $\widetilde{O}(m\sqrt{n})$ total update time. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 315–324, 2016. doi:10.1109/FOCS.2016.42.

10 Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *J. ACM*, 51(6):968–992, 2004. doi:10.1145/1039488.1039492.

11 Camil Demetrescu and Giuseppe F. Italiano. Mantaining dynamic matrices for fully dynamic transitive closure. *Algorithmica*, 51(4):387–427, 2008. doi:10.1007/s00453-007-9051-4.

12 Giuseppe Di Battista and Roberto Tamassia. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, 15(4):302–318, 1996. doi:10.1007/BF01961541.

13 Giuseppe Di Battista and Roberto Tamassia. On-line planarity testing. *SIAM J. Comput.*, 25(5):956–997, 1996. doi:10.1137/S0097539794280736.

14 Krzysztof Diks and Piotr Sankowski. Dynamic plane transitive closure. In *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 594–604, 2007. doi:10.1007/978-3-540-75520-3_53.

15 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997. doi:10.1145/265910.265914.

16 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification I: Planarity testing and minimum spanning trees. *J. Comput. Syst. Sci.*, 52(1):3–27, 1996. doi:10.1006/jcss.1996.0002.

17 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator-based sparsification II: Edge and vertex connectivity. *SIAM J. Comput.*, 28(1):341–381, 1998. Announced at STOC '93. doi:10.1137/S0097539794269072.

18 David Eppstein, Giuseppe F. Italiano, Roberto Tamassia, Robert Endre Tarjan, Jeffery Westbrook, and Moti Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algorithms*, 13(1):33–54, 1992. doi:10.1016/0196-6774(92)90004-V.

19 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. Syst. Sci.*, 72(5):868–889, 2006. doi:10.1016/j.jcss.2005.05.007.

20 Dora Giammarresi and Giuseppe F. Italiano. Decremental 2- and 3-connectivity on planar graphs. *Algorithmica*, 16(3):263–287, 1996. Announced at SWAT 1992. doi:10.1007/BF01955676.

21 Jens Gustedt. Efficient union-find for planar graphs and other sparse graph classes. *Theor. Comput. Sci.*, 203(1):123–141, 1998. doi:10.1016/S0304-3975(97)00291-0.

22 Carsten Gutwenger and Petra Mutzel. *A Linear Time Implementation of SPQR-Trees*, pages 77–90. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. doi:10.1007/3-540-44541-2_8.

23 Frank Harary. *Graph Theory.* Addison-Wesley Series in Mathematics. Addison Wesley, 1969.

24 Monika Rauch Henzinger and Mikkel Thorup. Sampling to provide or to bound: With applications to fully dynamic graph algorithms. *Random Struct. Algorithms*, 11(4):369–379, 1997. doi:10.1002/(SICI)1098-2418(199712)11:4<369::AID-RSA5>3.0.CO;2-X.

25 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001. doi:10.1145/502090.502095.

26 Jacob Holm, Giuseppe F Italiano, Adam Karczmarz, Jakub Lacki, Eva Rotenberg, and Piotr Sankowski. Contracting a planar graph efficiently. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 87. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

27 Jacob Holm, Giuseppe F. Italiano, Adam Karczmarz, Jakub Łącki, and Eva Rotenberg. Decremental SPQR-trees for Planar Graphs. *ArXiv e-prints*, 2018. arXiv:1806.10772.

28 Jacob Holm and Eva Rotenberg. Dynamic planar embeddings of dynamic graphs. *Theory of Computing Systems*, Apr 2017. doi:10.1007/s00224-017-9768-7.

29 Jacob Holm, Eva Rotenberg, and Christian Wulff-Nilsen. Faster fully-dynamic minimum spanning forest. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, Sept. 14-16, 2015, Proceedings*, pages 742–753, 2015. doi:10.1007/978-3-662-48350-3_62.

30 John E. Hopcroft and Robert Endre Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973. doi:10.1137/0202012.

31 John E. Hopcroft and Robert Endre Tarjan. A V log V algorithm for isomorphism of triconnected planar graphs. *J. Comput. Syst. Sci.*, 7(3):323–331, 1973. doi:10.1016/S0022-0000(73)80013-3.

32 John E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1974, Seattle, Washington, USA*, pages 172–184, 1974. doi:10.1145/800119.803896.

33 Giuseppe F. Italiano, Adam Karczmarz, Jakub Łącki, and Piotr Sankowski. Decremental single-source reachability in planar digraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1108–1121, 2017. doi:10.1145/3055399.3055480.

34 Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 313–322, 2011. doi:10.1145/1993636.1993679.

35 Goossen Kant. Algorithms for drawing planar graphs, 2001.

36 Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. Submatrix maximum queries in Monge matrices and Monge partial matrices, and their applications. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 338–355, 2012. URL: `http://portal.acm.org/citation.cfm?id=2095147&amp;CFID=63838676&amp;CFTOKEN=79617016`.

37 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1131–1142, 2013. `doi:10.1137/1.9781611973105.81`.

38 Casper Kejlberg-Rasmussen, Tsvi Kopelowitz, Seth Pettie, and Mikkel Thorup. Faster worst case deterministic dynamic connectivity. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 53:1–53:15, 2016. `doi:10.4230/LIPIcs.ESA.2016.53`.

39 Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 81–91, 1999. `doi:10.1109/SFFCS.1999.814580`.

40 Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, BC, Canada, January 23-25, 2005*, pages 146–155, 2005. URL: `http://dl.acm.org/citation.cfm?id=1070432.1070454`.

41 Philip N. Klein and Shay Mozes. Optimization algorithms for planar graphs, 2017. URL: `http://planarity.org`.

42 Jakub Łącki. Improved deterministic algorithms for decremental reachability and strongly connected components. *ACM Trans. Algorithms*, 9(3):27:1–27:15, 2013. `doi:10.1145/2483699.2483707`.

43 Jakub Łącki, Jakub Oćwieja, Marcin Pilipczuk, Piotr Sankowski, and Anna Zych. The power of dynamic distance oracles: Efficient dynamic algorithms for the Steiner tree. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 11–20, 2015. `doi:10.1145/2746539.2746615`.

44 Jakub Łącki and Piotr Sankowski. Min-cuts and shortest cycles in planar graphs in $O(n \log\log n)$ time. In *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, pages 155–166, 2011. `doi:10.1007/978-3-642-23719-5_14`.

45 Jakub Łącki and Piotr Sankowski. Optimal decremental connectivity in planar graphs. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 608–621, 2015. `doi:10.4230/LIPIcs.STACS.2015.608`.

46 Karl Menger. Zur allgemeinen kurventheorie. *Fund. Math.*, 10:96–115, 1927.

47 Danupon Nanongkai and Thatchaphol Saranurak. Dynamic spanning forest with worst-case update time: adaptive, Las Vegas, and O(n$^{1/2 - \epsilon}$)-time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1122–1129, 2017. `doi:10.1145/3055399.3055447`.

48 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *Proceedings of the 58th Annual Symposium on Foundations of Computer Science, FOCS 2017*, 2017. To appear.

49 Liam Roditty and Uri Zwick. Improved dynamic reachability algorithms for directed graphs. *SIAM J. Comput.*, 37(5):1455–1471, 2008. `doi:10.1137/060650271`.

50 Pierre Rosenstiehl. Embedding in the plane with orientation constraints: The angle graph. *Annals of the New York Academy of Sciences*, 555(1):340–346, 1989. doi:10.1111/j.1749-6632.1989.tb22470.x.

51 Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse (extended abstract). In *45th Symposium on Foundations of Computer Science FOCS 2004, 17-19 October 2004, Rome, Italy, Proceedings*, pages 509–517, 2004. doi:10.1109/FOCS.2004.25.

52 Shay Solomon. Fully dynamic maximal matching in constant update time. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 325–334, 2016. doi:10.1109/FOCS.2016.43.

53 Sairam Subramanian. A fully dynamic data structure for reachability in planar digraphs. In *Algorithms - ESA '93, First Annual European Symposium, Bad Honnef, Germany, September 30 - October 2, 1993, Proceedings*, pages 372–383, 1993. doi:10.1007/3-540-57273-2_72.

54 Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 343–350, 2000. doi:10.1145/335305.335345.

55 Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 112–119, 2005. doi:10.1145/1060590.1060607.

56 Christian Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1757–1769, 2013. doi:10.1137/1.9781611973105.126.

57 Christian Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1130–1143, 2017. doi:10.1145/3055399.3055415.

Computing the Chromatic Number Using Graph Decompositions via Matrix Rank

Bart M. P. Jansen[1]
Eindhoven University of Technology, Eindhoven, The Netherlands
b.m.p.jansen@tue.nl

Jesper Nederlof[2]
Eindhoven University of Technology, Eindhoven, The Netherlands
j.nederlof@tue.nl

Abstract

Computing the smallest number q such that the vertices of a given graph can be properly q-colored is one of the oldest and most fundamental problems in combinatorial optimization. The q-Coloring problem has been studied intensively using the framework of parameterized algorithmics, resulting in a very good understanding of the best-possible algorithms for several parameterizations based on the structure of the graph. For example, algorithms are known to solve the problem on graphs of treewidth tw in time $\mathcal{O}^*(q^{\mathrm{tw}})$, while a running time of $\mathcal{O}^*((q-\varepsilon)^{\mathrm{tw}})$ is impossible assuming the Strong Exponential Time Hypothesis (SETH). While there is an abundance of work for parameterizations based on decompositions of the graph by *vertex separators*, almost nothing is known about parameterizations based on *edge separators*. We fill this gap by studying q-Coloring parameterized by cutwidth, and parameterized by pathwidth in bounded-degree graphs. Our research uncovers interesting new ways to exploit small edge separators.

We present two algorithms for q-Coloring parameterized by cutwidth ctw: a deterministic one that runs in time $\mathcal{O}^*(2^{\omega\cdot\mathrm{ctw}})$, where ω is the matrix multiplication constant, and a randomized one with runtime $\mathcal{O}^*(2^{\mathrm{ctw}})$. In sharp contrast to earlier work, the running time is *independent* of q. The dependence on cutwidth is optimal: we prove that even 3-Coloring cannot be solved in $\mathcal{O}^*((2-\varepsilon)^{\mathrm{ctw}})$ time assuming SETH. Our algorithms rely on a new rank bound for a matrix that describes compatible colorings. Combined with a simple communication protocol for evaluating a product of two polynomials, this also yields an $\mathcal{O}^*((\lfloor d/2\rfloor+1)^{\mathrm{pw}})$ time randomized algorithm for q-Coloring on graphs of pathwidth pw and maximum degree d. Such a runtime was first obtained by Björklund, but only for graphs with few proper colorings. We also prove that this result is optimal in the sense that no $\mathcal{O}^*((\lfloor d/2\rfloor+1-\varepsilon)^{\mathrm{pw}})$-time algorithm exists assuming SETH.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized Complexity, Chromatic Number, Graph Decompositions

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.47

Related Version A full version is available at [28], `https://arxiv.org/abs/1806.10501`.

[1] NWO Veni grant "Frontiers in Parameterized Preprocessing" and NWO Gravitation grant "Networks"
[2] NWO Veni grant "Reducing small instances of complex tasks to large instances of simple ones" and NWO Gravitation grant "Networks"

© Bart M. P. Jansen and Jesper Nederlof;
licensed under Creative Commons License CC-BY

1 Introduction

Graph coloring is one of the most fundamental combinatorial problems, studied already in the 1850s. Countless papers (cf. [38]) and several monographs [29, 30, 33] have been devoted to its combinatorial and algorithmic investigation. Since the graph coloring problem is NP-complete even in restricted settings such as planar graphs [21], considerable effort has been invested in finding polynomial-time approximation algorithms and exact algorithms that beat brute-force search [5, 6].

A systematic study of which characteristics of inputs govern the complexity of the graph coloring problem has been undertaken using the framework of parameterized algorithmics. The aim in this framework is to obtain algorithms whose running time is of the form $f(k) \cdot n^{\mathcal{O}(1)}$, where k is a parameter that measures the complexity of the instance and is independent of the number of vertices n in the input graph. Over the past decade, numerous parameters have been employed that quantify the structure of the underlying graph. In several settings, algorithms have been obtained that are *optimal* under the Strong Exponential Time Hypothesis (SETH) [25, 26]. For example, it has long been known (cf. [10, Theorem 7.9],[40]) that testing q-colorability on a graph that is provided together with a tree decomposition of width k can be done in time $\mathcal{O}(q^k \cdot k^{\mathcal{O}(1)} \cdot n)$. Lokshtanov, Marx, and Saurabh [34] proved a matching lower bound: an algorithm running in time $(q-\varepsilon)^k \cdot n^{\mathcal{O}(1)}$ for any $\varepsilon > 0$ and integer $q \geq 3$ would contradict SETH. Results are also known for graph coloring parameterized by the vertex cover number [27], pathwidth and the feedback vertex number [34], cliquewidth [17, 24, 31], twin-cover [20], modular-width [19], and split-matching width [39]. (See [16, Fig. 1] for relations between these parameters.)

A survey of these algorithmic results for graph coloring results in the following picture of the complexity landscape: For graph parameters that are defined in terms of the width of decompositions by vertex separators (pathwidth, treewidth, vertex cover number, etc.), one can typically obtain a running time of $\mathcal{O}^*(q^k)$ to test whether a graph that is given together with a decomposition of width k is q-colorable, but assuming (S)ETH there is no algorithm with running time $\mathcal{O}^*(c^k)$ for any constant c independent of q [27, Theorem 11]. (We use $\mathcal{O}^*(f(k))$ as a shorthand for $f(k) \cdot n^{\mathcal{O}(1)}$.)

The complexity of graph coloring parameterized by width measures based on vertex separators is therefore well-understood by now. However, only little attention has been paid to graph decompositions whose width is measured in terms of the number of *edges* in a separator. There is intriguing evidence that separators consisting of few edges (or, equivalently, consisting of a bounded number of bounded-degree vertices) can be algorithmically exploited in nontrivial ways when solving q-COLORING. In 2016, Björklund [4] presented a fascinating algebraic algorithm that decides q-colorability using an algorithmic variation on the Alon-Tarsi theorem [1]. Given a graph G of maximum degree d, a path decomposition of width k, and integers q and s, his algorithm runs in time $(\lfloor d/2 \rfloor + 1)^k n^{\mathcal{O}(1)} \cdot s$. If the graph is not q-colorable it always outputs NO. If the graph has at most s proper q-colorings, then it outputs YES with constant probability. Hence when $q \geq (\lfloor d/2 \rfloor + 1)$ and s is small, it improves over the standard $\mathcal{O}^*(q^k)$-time dynamic program by exploiting the bounded-degree vertex separators encoded in the path decomposition. However, the dependence of the running time on the number of proper q-colorings in the graph is very undesirable, as that number may be exponentially large in n.

Björklund's algorithm hints at the fact that graph decompositions whose width is governed by the number of *edges* in a separator may yield an algorithmic advantage over existing approaches. In this work, we therefore perform a deeper investigation of how decompositions by small edge separators can be exploited when solving q-Coloring. By leveraging a new rank upper bound for a matrix that describes the compatibility of colorings of subgraphs on two sides of a small edge separator, we obtain a number of novel algorithmic results. In particular, we show how to eliminate dependence on the number s of proper colorings.

Our results. We present efficient algorithms for q-Coloring parameterized by the width of various types of graph decompositions by small edge separators. Our first results are phrased in terms of the graph parameter *cutwidth*. A decomposition in this case corresponds to a linear ordering of the vertices; the cutwidth of this ordering is given by the maximum number of edges that connect a vertex in a prefix of the ordering to a vertex in the complement (see Section 2 for formal definitions). Cutwidth is one of the classic graph layout parameters (cf. [14]). It takes larger values than treewidth [32], and has been the subject of frequent study [23, 41, 42].

Informally speaking, we prove that interactions of partial solutions on low-cutwidth graphs are much simpler than interactions of partial solutions on low-pathwidth graphs. The rank-based approach developed in earlier work [8, 11, 18] can be used by setting up matrices whose rank determines the complexity of these interactions in low-cutwidth graphs. These are different from the matrices associated to partial solutions in low-pathwidth graphs, and admit better rank bounds. This is exploited by two different algorithms: a deterministic algorithm that employs fast matrix multiplication and therefore has the matrix-multiplication constant ω in its running time, and a faster randomized Monte Carlo algorithm.

▶ **Theorem 1.** *There is a deterministic algorithm that, for any q, solves q-Coloring on a graph G with a given linear layout of cutwidth* ctw *in $\mathcal{O}^*(2^{\omega \cdot \mathrm{ctw}})$ time, where $\omega \leq 2.373$ is the matrix multiplication constant.*

▶ **Theorem 2.** *There is a randomized Monte Carlo algorithm that, for any q, solves q-Coloring on a graph G with a given linear layout of cutwidth* ctw *in $\mathcal{O}^*(2^{\mathrm{ctw}})$ time.*

These results show a striking difference between cutwidth and parameterizations based on vertex separators such as treewidth and vertex cover number: we obtain single-exponential running times where the base of the exponent is *independent* of the number of colors q, which (assuming ETH) is impossible even parameterized by vertex cover [27]. The assumption that a decomposition is given in the input is standard in this line of research [8, 12, 11, 18] and decouples the complexity of *finding* a decomposition from that of *exploiting* a decomposition.

The ideas underlying Theorems 1 and 2 can also be used to eliminate the dependence on the number of proper colorings from Björklund's algorithm. We prove the following theorem:

▶ **Theorem 3.** *There is a randomized Monte Carlo algorithm that, for any q, solves q-Coloring on a graph G with maximum degree d and given path decomposition of width* pw *in $\mathcal{O}^*((\lfloor d/2 \rfloor + 1)^{\mathrm{pw}})$ time.*

Our approach uses the first step of the proof of the Alon-Tarsi theorem (i.e. rewrite the problem into evaluating the graph polynomial) and also relates colorability to certain orientations, but deviates from the previous algorithm otherwise: to evaluate the appropriate graph polynomial we extend a fairly simple communication-efficient protocol to evaluate a product of two polynomials.

We also prove that the randomized algorithms of Theorem 2 and Theorem 3 are conditionally *optimal*, even when restricted to special cases:

▶ **Theorem 4 (★).** *Assuming SETH, there is no $\varepsilon > 0$ such that 3-Coloring on a planar graph G given along with a linear layout of cutwidth* ctw *can be solved in time $\mathcal{O}^*((2-\varepsilon)^{\mathrm{ctw}})$.*

▶ **Theorem 5** (★). *Let $d \geq 5$ be an odd integer and let $q_d := \lfloor d/2 \rfloor + 1$. Assuming SETH, there is no $\varepsilon > 0$ such that q_d-*Coloring *on a graph of maximum degree d given along with a path decomposition of pathwidth* pw *can be solved in time $\mathcal{O}^*((\lfloor d/2 \rfloor + 1 - \varepsilon)^{\mathrm{pw}})$.*

These results are obtained by building on the techniques of Lokshtanov et al. [34] that propagate 'partial assignments' throughout graphs of small cutwidth or pathwidth.

Organization

In Section 2 we provide preliminaries. In Section 3 we present algorithms for graph coloring, proving Theorems 1, 2, and 3. In Section 4 we give briefly sketch the main ideas of the proofs of Theorems 4 and 5, showing that our randomized algorithms cannot be improved significantly assuming SETH. Finally, we provide some conclusions in Section 5. Due to space restrictions, proofs for statements marked (★) have been deferred to the full version [28].

2 Preliminaries

We use $\mathbb{N}$ to denote the natural numbers, including 0. For a positive integer n and a set X we use $\binom{X}{n}$ to denote the collection of all subsets of X of size n. The *power set* of X is denoted 2^X. The set $\{1, \ldots, n\}$ is abbreviated as $[n]$. The $\mathcal{O}^*$ notation suppresses polynomial factors in the input size n, such that $\mathcal{O}^*(f(k))$ is shorthand for $\mathcal{O}(f(k)n^{\mathcal{O}(1)})$. All our logarithms have base two. For sets S, T we denote by S^T the set of vectors indexed by elements of T whose entries are from S. If $T = [n]$, we use S^n instead of $S^{[n]}$.

We consider finite, simple, and undirected graphs G, consisting of a vertex set $V(G)$ and edge set $E(G) \subseteq \binom{V(G)}{2}$. The neighbors of a vertex v in G are denoted $N_G(v)$. The closed neighborhood of v is $N_G[v] := N_G(v) \cup \{v\}$. The degree $d(v)$ equals $|N_G(v)|$ and if $X \subseteq E(G)$, then $d_X(v)$ denotes the number of edges of X incident to v. This notation is extended to $d^-(v), d^+(v), d^-_X(v), d^+_X(v)$ for directed graphs in the natural way (e.g. $d^+_X(v)$ denotes the number of w such that $(v, w) \in X$). For a vertex set $S \subseteq V(G)$ the open neighborhood is $N_G(S) := \bigcup_{v \in S} N_G(v) \setminus S$ and the closed neighborhood is $N_G[S] := N_G(S) \cup S$, while $G[S]$ denotes the graph induced by S.

A q-coloring of a graph G is a function $f \colon V(G) \to [q]$. A coloring is *proper* if $f(u) \neq f(v)$ for all edges $\{u, v\} \in E(G)$. For a fixed integer q, the q-Coloring problem asks whether a given graph G has a proper q-coloring. The q-SAT problem asks whether a given Boolean formula, in conjunctive normal form with clauses of size at most q, has a satisfying assignment.

▶ **Strong Exponential Time Hypothesis** ([25, 26]). *For every $\varepsilon > 0$, there is a constant q such that q-*SAT *on n variables cannot be solved in time $\mathcal{O}^*((2 - \varepsilon)^n)$.*

Cutwidth. For an n-vertex graph G, a *linear layout* of G is a linear ordering of its vertex set, given by a bijection $\pi \colon V(G) \to [n]$. The *cutwidth* of G with respect to the layout π is:

$$\mathrm{ctw}_\pi(G) = \max_{1 \leq i < n} \left| \left\{ \{u, v\} \in E(G) \;\middle|\; \pi(u) \leq i \wedge \pi(v) > i \right\} \right|,$$

and the cutwidth $\mathrm{ctw}(G)$ of a graph G is the minimum cutwidth attained by any linear layout. It is well-known (cf. [7]) that $\mathrm{ctw}(G) \geq \mathrm{pw}(G) \geq \mathrm{tw}(G)$, where the latter denote the pathwidth and treewidth of G, respectively. An intuitive way to think about cutwidth is to consider the vertices as being placed on a horizontal line in the order dictated by the layout π, with edges drawn as x-monotone curves. For any position i we consider the gap between vertex $\pi^{-1}(i)$ and $\pi^{-1}(i+1)$, and count the edges that *cross* the gap by having one endpoint at position at most i and the other at position after i. The cutwidth of a layout is the maximum number of edges crossing any single gap.

Pathwidth and path decompositions. A *path decomposition* of a graph G is a path P in which each node x has an associated set of vertices $B_x \subseteq V(G)$ (called a *bag*) such that $\bigcup_{x \in V(P)} B_x = V(G)$ and the following properties hold:

1. For each edge $\{u, v\} \in E(G)$ there is a node x in P such that $u, v \in B_x$.
2. If $v \in B_x \cap B_y$ then $v \in B_z$ for all nodes z on the (unique) path from x to y in P.

The *width* of P is the size of the largest bag minus one, and the pathwidth of a graph G is the minimum width over all possible path decompositions of G. Since our focus here is on dynamic programming over a path decomposition we only mention in passing that the related notion of treewidth can be defined in the same way, except for letting the nodes of the decomposition form a tree instead of a path.

It is common for the presentation of dynamic-programming algorithms to use path- and tree decompositions that are normalized in order to make the description easier to follow. For an overview of tree decompositions and dynamic programming on tree decompositions see e.g. [9]. Following [12] we use the following path decompositions:

▶ **Definition 6** (Nice Path Decomposition)**.** A *nice path decomposition* is a path decomposition where the underlying path of nodes is ordered from left to right (the predecessor of any node is its left neighbor) and in which each bag is of one of the following types:

- **First (leftmost) bag**: the bag associated with the leftmost node x is empty, $B_x = \emptyset$.
- **Introduce vertex bag**: an internal node x of P with predecessor y such that $B_x = B_y \cup \{v\}$ for some $v \notin B_y$. This bag is said to *introduce* v.
- **Introduce edge bag**: an internal node x of P labeled with an edge $\{u, v\} \in E(G)$ with one predecessor y for which $u, v \in B_x = B_y$. This bag is said to *introduce* $\{u, v\}$.
- **Forget bag**: an internal node x of P with one predecessor y for which $B_x = B_y \setminus \{v\}$ for some $v \in B_y$. This bag is said to *forget* v.
- **Last (rightmost) bag**: the bag associated with the rightmost node x is empty, $B_x = \emptyset$.

It is easy to verify that any given path decomposition of pathwidth pw can be transformed in time $|V(G)| \cdot \mathrm{pw}^{\mathcal{O}(1)}$ into a nice path decomposition without increasing the width. Let $B_1, \ldots, B_\ell$ be a nice path decomposition of G. We say B_i is *before* B_j if $i \leq j$. We denote $V_i = \bigcup_{j=1}^{i} B_i$ and let E_i denote the set of edges introduced in bags before i.

3 Upper bounds for Graph Coloring

In this section we outline algorithms for q-Coloring that run efficiently when given a graph and either a small-cutwidth layout or a good path decomposition on graphs with small maximum degree. We assume the input graph has no isolated vertices, as they are clearly irrelevant. We start by using the 'rank-based approach' as proposed in [8] to obtain deterministic algorithms, and afterward give a randomized algorithm with substantial speedup. In both approaches the idea is to employ dynamic programming to accumulate needed information about the existence of partial solutions, but use linear-algebraic methods to compress this information. Let us remark in passing that our approaches are robust in the sense that they directly extend to generalizations such as q-List Coloring in which for every vertex a set of allowed colors is given.[3]

A key quantity that determines the amount of information needed after compression in general is the rank of a *partial solutions matrix.* This matrix has its rows and columns

[3] In the deterministic approach we simply avoid partial solutions not satisfying these constraints, and in the randomized approach we assign sufficiently large weight to disallowed (vertex,color) combinations.

indexed by partial solutions (which could be defined in various ways) and an entry is 1 (or more generally, non-zero) if the two partial solutions combine to a solution. Previously, this method proved to be highly useful for connectivity problems parameterized by treewidth [8]. For q-Coloring parameterized by treewidth, partial solutions can naturally be defined as partial proper colorings of a subgraph whose boundary is formed by some vertex separator. Two partial colorings combine to a proper complete coloring if and only if the two partial colorings agree on the coloring of the separator. Unfortunately, the rank-based approach is not useful here as the partial solution matrices arising have large rank, as witnessed by induced identity submatrices of dimensions q^{tw}. Indeed, the lower bound under SETH by Lokshtanov, Marx, and Saurabh [34] shows that no algorithm can solve the problem much faster than $\mathcal{O}^*(q^{\mathrm{pw}})$, where pw denotes the pathwidth of the input graph.

Still, this does not exclude much faster running times parameterized by *cutwidth.* In our application of the rank-based approach for q-Coloring of a graph with a given linear layout of cutwidth ctw, the partial solutions are q-colorings of the first i and last $n-i$ vertices in the linear order, and clearly only the colors assigned to vertices incident to the edges going over the cut are relevant. If we let $X = X_i, Y = Y_i$ denote the endpoints of these edges occurring respectively not after and after i, and let $H = H_i$ denote the bipartite graph induced by the cut and these edges, we are set to study the rank of the following partial solutions matrix indexed by $x \in [q]^X$ and $y \in [q]^Y$:

$$M_H[x,y] = \begin{cases} 1, & \text{if } x \cup y \text{ is a proper } q\text{-coloring of } H, \\ 0, & \text{if otherwise.} \end{cases}$$

Here and below, we slightly abuse notation by viewing elements of V^I (i.e. vectors with values in V that are indexed by I) as sets of pairs in $I \times V$; that is, if $x \in V^I$ we also use x to denote the set $\{(i, x_i)\}_{i \in I}$. With this notation in mind, note that $x \cup y$ above can be interpreted as an element of $[q]^{X \cup Y}$ in the natural way as X and Y are disjoint. As the rank of M_H is generally high[4] and depends on q, we instead focus on the matrix M'_H defined by

$$M'_H[x,y] = \prod_{(v,w) \in E(H)} (x_v - y_w), \tag{1}$$

where all edges are directed from X to Y in $E(H)$. The crux is that the support (e.g. the set of non-zero entries) of M'_H equals the support of M_H:

▶ **Lemma 7.** *We have $M'_H[x,y] \neq 0$ if and only if $x \cup y$ is a proper q-coloring of H.*

Proof. If $x_v = y_w$ for some $(v,w) \in E(H)$ then the term $(x_v - y_w)$ is zero, implying the entire product on the right hand-side of (1) is zero. If x and y differ at every coordinate, then $M'_H[x,y]$ is a product of nonzero terms, and therefore non-zero itself. ◀

In Sections 3.1–3.2 this property will allow us to work with M'_H instead of M_H, when combined with the Isolation Lemma or Gaussian-elimination approach; similarly as in previous work [8, 11, 12].[5]

[4] For example, if H is a single edge M_H is the complement of an identity matrix of dimensions $q \times q$.
[5] In the deterministic setting, the observation that one can work with a matrix different from a partial solution matrix but with the same support as the partial solution matrix was already used by Fomin et al. [18] in combination with a matrix factorization by Lovász [36].

3.1 A deterministic algorithm

We first show that M'_H has rank at most $\prod_{v\in X}(d_{E(H)}(v)+1)$ by exhibiting an explicit factorization. Here we use the shorthand $d_W(v)$ for the number of edges in W containing vertex v. For a bipartite graph H with parts X, Y and edges oriented from X to Y, we have:

$$\begin{aligned} M'_H[x,y] &= \prod_{(v,w)\in E(H)} (x_v - y_w) \\ &= \sum_{W\subseteq E(H)} \left(\prod_{v\in X} x_v^{d_W(v)}\right)\left(\prod_{v\in Y}(-y_v)^{d_{E(H)\setminus W}(v)}\right) \\ &= \sum_{(d_v\in\{0,\dots,d_{E(H)}(v)\})_{v\in X}} \left(\prod_{v\in X} x_v^{d_v}\right)\left(\sum_{\substack{W\subseteq E(H)\\ \forall v\in X: d_W(v)=d_v}} \prod_{v\in Y}(-y_v)^{d_{E(H)\setminus W}(v)}\right), \end{aligned} \quad (2)$$

where the second equality follows by expanding the product and the third equality follows by grouping the summands on the number of edges incident to vertices in W included in X.

Expression (2) provides us with a matrix factorization $M'_H = L_H \cdot R_H$ where L_H is indexed by $x \in [q]^X$ and a sequence $s = (d_v \in \{0,\dots,d_{E(H)}(v)\})_{v\in X}$ and R_H has columns indexed by $y \in [q]^Y$ (one such factorization sets $L_H[x,s] = \prod_{v\in X} x_v^{s_v}$). As the number of relevant sequences s is bounded by $\prod_{v\in X}(d_{E(H)}(v)+1)$, the factorization implies the claimed rank bound for M'_H.[6] The rank bound allows some partial solutions to be pruned from the dynamic-programming table without changing the answer. The following definition captures correct reduction steps.

▶ **Definition 8.** Fix a bipartite graph H with parts X and Y and let $\mathcal{S} \subseteq [q]^X$ be a set of q-colorings of X. We say $\mathcal{S}' \subseteq [q]^X$ *H-represents* $\mathcal{S}$ if $\mathcal{S}' \subseteq \mathcal{S}$, and for each $y \in [q]^Y$ we have:

$$(\exists x \in \mathcal{S}\colon x \cup y \text{ is a proper coloring of } H) \Leftrightarrow (\exists x' \in \mathcal{S}'\colon x' \cup y \text{ is a proper coloring of } H). \quad (3)$$

Note that the backward direction of (3) is implied by the property that $\mathcal{S}' \subseteq \mathcal{S}$, but we state both for clarity. If H is clear from context it will be omitted. For future reference we record the observation that the transitivity of this relation follows directly from its definition:

▶ **Observation 9.** *Let H be a bipartite graph with parts X and Y, and let $\mathcal{A}, \mathcal{B}, \mathcal{C} \subseteq [q]^X$. If $\mathcal{A}$ represents $\mathcal{B}$ and $\mathcal{B}$ represents $\mathcal{C}$, then $\mathcal{A}$ represents $\mathcal{C}$.*

Given the above matrix factorization, we can directly follow the proof of [8, Theorem 3.7] to get the following result (note that ω denotes the matrix multiplication constant):

▶ **Lemma 10.** *There is an algorithm* `reduce` *that, given a bipartite graph H with parts X, Y and a set $\mathcal{S} \subseteq [q]^X$, outputs in time $\left(\prod_{v\in X}(d_{E(H)}(v)+1)\right)^{\omega-1} \cdot |\mathcal{S}| \cdot \mathrm{poly}(|X|+|Y|)$ a set $\mathcal{S}'$ that represents $\mathcal{S}$ and satisfies $|\mathcal{S}'| \leq \prod_{v\in X}(d_{E(H)}(v)+1)$.*

Proof. The algorithm is as follows: compute explicitly the matrix $L_H[\mathcal{S}, \cdot]$ (i.e. the submatrix of L_H induced by all rows in $\mathcal{S}$). As every entry of L_H can be computed in polynomial time, clearly this can be done within the claimed time bound. Subsequently, the algorithm finds a row basis of this matrix and returns that set as $\mathcal{S}'$. As the rank of a matrix is at most its

[6] This construction (first developed in this paper) has subsequently been used by the second author with Bansal et al. [3] in the completely different setting of online algorithms; see [3, Footnote 3].

number of columns, $|\mathcal{S}'| \leq \prod_{v \in X}(d_{E(H)}(v)+1)$. Using [8, Lemma 3.15], this step also runs in the promised running time.

To see that $\mathcal{S}'$ represents $\mathcal{S}$, note that clearly $\mathcal{S}' \subseteq \mathcal{S}$ and thus it remains to prove the forward implication of (3). To this end, suppose that $x \cup y$ is a proper q-coloring of H and $x \in \mathcal{S}$. As $\mathcal{S}'$ is a row basis of L_H, there exist $x^{(1)}, \dots, x^{(\ell)} \in \mathcal{S}'$ and $\lambda_1, \dots, \lambda_\ell$ such that

$$M'_H[x,y] = L_H[x,\cdot]R_H[\cdot,y] = \left(\sum_{i=1}^{\ell} \lambda_i L_H[x^{(i)},\cdot]\right) R_H[\cdot,y] = \sum_{i=1}^{\ell} \lambda_i M'_H[x^{(i)},y],$$

where $L_H[x,\cdot]$ and $R_H[\cdot,y]$ denote a row of L_H and column of R_H respectively. As $x \cup y$ is a proper coloring of H, Lemma 7 implies $M'_H[x,y]$ is non-zero. Therefore there must also exist $x^{(i)} \in \mathcal{S}'$ such that $M'_H[x^{(i)},y]$ is non-zero and hence $x^{(i)} \cup y$ is a proper coloring of H. ◀

Equipped with the algorithm `reduce` from Lemma 10 we are ready to present the algorithm for q-Coloring. On a high level, the algorithm uses a naïve dynamic-programming scheme, but by extensive use of the `reduce` procedure we efficiently represent sets of partial solutions and speed up the computation significantly.

First we need to introduce some notation. A vector $x \in V^I$ is *an extension* of a vector $x' \in V^{I'}$ if $I' \subseteq I$ and $x'_i = x_i$ for every $i \in I'$. If $x \in V^I$ and $P \subseteq I$ then the projection $x_{|P}$ is defined as the unique vector in V^P of which x is an extension. Let G be the graph for which we need to decide whether a proper q-coloring exists and fix an ordering $v_1, \dots, v_n$ of $V(G)$. We denote all edges as directed pairs (v_i, v_j) with $i < j$. For $i = 1, \dots, n$, define V_i as the i'th prefix of this ordering, C_i as the i'th cut in this ordering, and X_i and Y_i as the left and respectively right endpoints of the edges in this cut, i.e.

$$\begin{aligned} V_i &= \{v_1, \dots, v_i\}, & C_i &= \{(v_l, v_r) \in E(G) : l \leq i < r\}, \\ X_i &= \{v_l \in V(G) : \exists (v_l, v_r) \in C_i \wedge l < r\}, & Y_i &= \{v_r \in V(G) : \exists (v_l, v_r) \in C_i \wedge l < r\}. \end{aligned}$$

Note that $X_i \subseteq X_{i-1} \cup \{v_i\}$ and $Y_{i-1} \subseteq Y_i \cup \{v_i\}$. We let H_i denote the bipartite graph with parts X_i, Y_i and edge set C_i. For $i = 1, \dots, n$, let $T[i] \subseteq [q]^{X_i}$ be the set of all q-colorings of the vertices in X_i that can be extended to a proper q-coloring of $G[V_i]$. The following lemma shows that we can continuously work with a table T' that represents a table T:

▶ **Lemma 11.** *If $T'[i-1]$ H_{i-1}-represents $T[i-1]$, then $T'[i]$ H_i-represents $T[i]$, where*

$$T'[i] = \left\{ (x \cup (v_i, c))_{|X_i} : x \in T'[i-1], c \in [q], \left(\forall v \in N(v_i) \cap X_{i-1} \colon x_v \neq c\right) \right\}. \tag{4}$$

Proof. Assuming the hypothesis, we first show that $T'[i] \subseteq T[i]$. Let $x \in T'[i-1]$ and $c \in [q]$ such that $\forall v \in N(v_i) \cap X_{i-1} \colon x_v \neq c$. As $T'[i-1]$ represents $T[i-1]$, we have that $x \in T[i-1]$. By definition of $T[i-1]$, there exists a proper coloring w of $G[V_{i-1}]$ that extends x. Since all $v \in N(v_i) \cap X_{i-1} = N(v_i) \cap V_{i-1}$ satisfy $x_v \neq c$, it follows that $w \cup (v_i, c)$ is a proper coloring of $G[V_i]$, and thus $(x \cup (v_i, c))_{|X_i} \in T[i]$.

Thus, to prove the lemma it remains to show the forward implication of (3). To this end, let $x \in T[i]$ and let $w \in [q]^{V_i}$ be a proper coloring of $G[V_i]$ that extends x. Let $y \in [q]^{Y_i}$ be such that $x \cup y$ is a proper coloring of H_i. As $w_{v_i} \neq w_{v_j}$ for neighbors $v_j \in N(v_i) \cap V_{i-1}$ and $w_{v_i} \neq y_{v_j}$ for $v_j \in N(v_i) \setminus V_i$, it follows that $w \cup y$ extends a proper coloring of H_{i-1}.

Therefore $w_{|X_{i-1}} \cup (y \cup (v_i, w_{v_i}))_{|Y_{i-1}}$ must be a proper coloring of H_{i-1}, and $w_{|X_{i-1}} \in T[i-1]$ as it can be extended to a proper coloring of V_i, and thus also to a proper coloring of V_{i-1}. As $T'[i-1]$ H_{i-1}-represents $T[i-1]$, there exists $x' \in T'[i-1]$ such that $x' \cup (y \cup (v_i, w_{v_i}))_{|Y_{i-1}}$ is a proper coloring of H_{i-1}.

As no neighbor of v_i was assigned color w_{v_i} by y, it follows that $(x' \cup (v_i, w_{v_i})) \cup y$ is an extension of a proper coloring of H_i. As $x' \cup (y \cup (v, w_{v_i}))_{|Y_{i-1}}$ is a proper coloring of H_{i-1}, no neighbors of v_i are assigned color w_{v_i} by x', and by (4) we have that $(x' \cup (v_i, w_{v_i})) \in T'[i]$, as required. ◀

Now we combine Lemma 10 with Lemma 11 to obtain an algorithm to solve q-Coloring.

▶ **Lemma 12.** *q-Coloring can be solved in time $\mathcal{O}^*\left(\left(\max_i \prod_{v \in X_i}(d_{E(H_i)}(v)+1)\right)^{\omega}\right)$.*

Proof. Note $T'[0] = T[0] = \{\emptyset\}$ (where $\emptyset$ is the 0-dimensional vector). Using Lemma 11, we can use (4) for $i = 1, \ldots, n$ to iteratively compute a set $T'[i]$ representing $T[i]$ from a set $T'[i-1]$ representing $T[i-1]$, and replace $T'[i]$ after each step with $\texttt{reduce}(H_i, T'[i])$. By combining Lemma 11 and Observation 9, we may conclude that G has a q-coloring if and only if $T'[n]$ is not empty (that is, it contains a single element which is the empty vector).

The time required for the computation dictated by (4) is clearly $|T'[i]| \cdot \mathrm{poly}(n)$. Since $|T'[i-1]| \leq \max_i \prod_{v \in X_i}(d_{E(H_i)}(v)+1)$, as it is the result of `reduce`, we have that $|T'[i]|$ is bounded by $q \cdot \max_i \prod_{v \in X_i}(d_{E(H_i)}(v)+1)$. Using this upper bound for $T'[i]$, the time of `reduce` will be $\mathcal{O}^*\left(\left(\max_i \prod_{v \in X_i}(d_{E(H_i)}(v)+1)\right)^{\omega}\right)$, which clearly is the bottleneck in the running time. ◀

Theorem 1 now follows directly from this more general statement.

Proof of Theorem 1. If $v_1, \ldots, v_n$ is a layout of cutwidth k, then $|E(H_i)| \leq k$ for every i, and the term $\prod_{v \in X_i}(d_{E(H_i)}(v)+1)$ is upper bounded by 2^k by the AM-GM inequality. Thus the theorem follows from Lemma 12. ◀

3.2 A randomized algorithm

In this section we use an idea similar to the idea from the matrix factorization of the previous section to obtain faster randomized algorithms. Specifically, our main technical result is as follows (recall that E_i denotes the set of edges introduced in bags before B_i).

▶ **Theorem 13.** *There is a Monte Carlo algorithm for q-Coloring that, given a graph G and a nice path decomposition $B_1, \ldots, B_\ell$, runs in time $\mathcal{O}^*(\max_i \prod_{v \in B_i}(\min\{d_{E_i}(v), d(v) - d_{E_i}(v)\}+1))$. The algorithm does not give false-positives and returns the correct answer with high probability.*

Let $V(G) = V = \{v_1, \ldots, v_n\}$ be ordered arbitrarily, and direct every edge $\{v_i, v_j\}$ as (v_i, v_j) with $i < j$. Define the *graph polynomial* f_G as $f_G(x_1, \ldots, x_n) = \prod_{(u,v) \in E(G)}(x_u - x_v)$. This polynomial has been studied intensively (cf. [2, 13, 35]), for example in the context of the Alon-Tarsi theorem [1]. Define $P_G = \sum_{x \in [q]^V} f_G(x)$. Similarly as in Lemma 7 we see that if $P_G \neq 0$ then G has a proper q-coloring, and if G has a unique q-coloring then $P_G \neq 0$ as it is the product of non-zero values. This is useful if the graph is guaranteed to have at most one proper q-coloring. To this end, we use a standard technique based on the Isolation Lemma, which we state now.

▶ **Definition 14.** A function $\omega\colon U \to \mathbb{Z}$ *isolates* a set family $\mathcal{F} \subseteq 2^U$ if there is a unique $S' \in \mathcal{F}$ with $\omega(S') = \min_{S \in \mathcal{F}} \omega(S)$, where $\omega(S') := \sum_{v \in S'} \omega(v)$.

▶ **Lemma 15** (Isolation Lemma, [37]). *Let $\mathcal{F} \subseteq 2^U$ be a non-empty set family over universe U. For each $u \in U$, choose a weight $\omega(u) \in \{1, 2, \ldots, W\}$ uniformly and independently at random. Then* $\Pr[\omega \text{ isolates } \mathcal{F}] \geq 1 - |U|/W$.

We will apply Lemma 15 to isolate the set of proper colorings of G. To this end, we use the set $V(G) \times [q]$ of vertex/color pairs as our universe U, and consider a weight function $\omega\colon V(G) \times [q] \to \mathbb{Z}$.

▶ **Definition 16.** A *q-coloring* of G is a vector $x \in [q]^n$, and it is proper if $x_i \neq x_j$ for every $(i,j) \in E(G)$. The *weight* of x is $\omega(x) = \sum_{i=1}^n \omega((i, x_i))$.

Let $\omega\colon V(G) \times [q] \to [2nq]$ be a random weight function, i.e. for every $v \in V(G)$ and $c \in [q]$ we pick an integer from $[2nq]$ uniformly and independently at random. For every integer z we associate a number $P_G(z)$ with G, as follows:

$$P_G(z) = \sum_{\substack{x \in [q]^n \\ \omega(x)=z}} \prod_{(i,j) \in E(G)} (x_i - x_j)\,. \tag{5}$$

If G has no proper q-coloring, then $P_G(z) = 0$ since for every q-coloring x there will be an edge $(i,j) \in E$ for which $x_i = x_j$ and therefore the product in (5) vanishes. We claim that if G has a proper q-coloring, then with probability at least $1/2$ there exists $z \leq 2nq$ such that $P_G(z) \neq 0$, which means we get a correct algorithm with high probability by repeating a polynomial in n number of times. Let $\mathcal{F} = \{\{(i,x_i)\}_{i \in V} : x \text{ is a proper } q\text{-coloring of } G\} \subseteq 2^U$. As $\mathcal{F}$ is non-empty, we may apply Lemma 15 to obtain that ω isolates $\mathcal{F}$ with probability at least $1/2$. Conditioned on this event, there must exist an integer w such that there is exactly one proper q-coloring x of G satisfying $\omega(x) = z$. In this case, x is the only summand in (5) that can have a non-zero contribution. Moreover, as it is a proper coloring, its contribution is a product of non-zero entries and therefore non-zero itself. Thus $P_G(z)$ is non-zero with probability at least $1/2$.

We now continue by showing how to compute $P_G(z)$ for all $z \leq 2nq$ quickly using dynamic programming. Note that by expanding the product in (5) we have:

$$P_G(z) = \sum_{\substack{x \in [q]^n \\ \omega(x)=z}} \sum_{W \subseteq E(G)} \left(\prod_{(u,v) \in W} x_u \right) \left(\prod_{(u,v) \in E(G) \setminus W} -x_v \right). \tag{6}$$

If B_i is a bag of a path decomposition (Section 2), we need to define table entries T_i containing all information about the graph (V_i, E_i) needed to compute $P_G(z)$. Before we describe these table entries we make a small deviation to convey intuition about our approach. Specifically, we may interpret $P_G(z)$ as a polynomial in variables x_v for $v \in B_i$. Now suppose for simplicity that $|B_i| = 1$. Then the amount of information about E_i needed to compute $P_G(z)$ may be studied via a simple communication-complexity game that we now outline.

A One-way Communication Protocol. Alice has a univariate polynomial $P_A(x)$ of degree d_A, and Bob has a univariate polynomial $P_B(x)$ of degree d_B. Both parties know d_A, d_B and an additional integer q. Alice needs to send as few bits as possible to Bob after which Bob needs to output the quantity $\sum_{x \in [q]} P_A(x) P_B(x)$, where $q \in \mathbb{N}$ is known to both.

An easy strategy is that Alice sends the $d_A + 1$ coefficients of her polynomial to Bob. An alternative strategy for Alice is based on partial evaluations, which is useful when $d_B < d_A$. By expanding Bob's polynomial in coefficient form we can rewrite $\sum_{x \in [q]} P_A(x) P_B(x)$ into

$$\sum_{x \in [q]} P_A(x)(c_0 x^0 + c_1 x^1 + \ldots + c_{d_B} x^{d_B}) = c_0 \sum_{x \in [q]} P_A(x) x^0 + \ldots + c_{d_B} \sum_{x \in [q]} P_A(x) x^{d_B},$$

so as second strategy Alice may send the $d_B + 1$ values $\sum_{x\in[q]} P_A(x)x^i$ for $i = 0, \dots, d^B$. So she can always send at most $\min\{d_A, d_B\} + 1$ integers.

In our setting for defining table entries T_i for evaluating $P_G(z)$, we think of $d_A(v)$ as the number of edges in E_i incident to v and of $d_B(v)$ as the number of edges incident to v not in E_i. Roughly speaking, the running time of Theorem 13 is obtained by defining table entries storing Alice's message, in which she chooses the best of the two strategies independently for every vertex.

Definition of the Table Entries. An *orientation* O of a subset $X \subseteq E(G)$ of edges is a set of directed pairs such that for every $\{u, v\} \in X$, either $(u, v) \in O$ or $(v, u) \in O$. If O is an orientation of X, we also say O *orients* X. The number of *reversals* $\mathrm{rev}(O)$ of O is the number of $(v, u) \in O$ such that u is introduced in a bag before the bag in which v is introduced. An orientation is *even* if its number of reversals is even, and it is *odd* otherwise.

For a fixed path decomposition $B_1, \dots, B_\ell$ of the input graph G, let $L_i \subseteq B_i$ consist of all vertices in B_i of which at most half of their incident edges are already introduced in B_i or a bag before B_i, and let $R_i = B_i \setminus L_i$. Let l^i be the vector indexed by L_i such that for every $v \in L_i$ the value l^i_v denotes the number of edges incident to v already introduced before or at bag B_i. Similarly, let r^i be the vector indexed by R_i such that for every $v \in R_i$ the value r^i_v denotes the number of edges incident to v introduced *after* bag B_i. So for every i we have $d(v) = l^i_v + r^i_v$.

If $b \in \mathbb{N}^I_{\geq 0}$ is a vector, we denote $\mathcal{P}(b)$ for the set of vectors a in $\mathbb{N}^I_{\geq 0}$ such that $a \preceq b$. Here $a \preceq b$ denotes that $a_v \leq b_v$ for every $v \in I$. For $d \in \mathcal{P}(l^i)$ and $e \in \mathcal{P}(r^i)$, define:

$$T^z_i[d, e] = \sum_{\substack{x\in[q]^{V_i\setminus L_i}\\ \omega(x)=z}} \; \sum_{\substack{O \text{ orients } E_i\\ \forall u\in L_i: d^+_O(u)=d_u}} (-1)^{\mathrm{rev}(O)} \left(\prod_{u\in V_i\setminus L_i} x_u^{d^+_O(u)}\right)\left(\prod_{u\in R_i} x_u^{e_u}\right). \tag{7}$$

Intuitively, this could be seen as a partial evaluation of $P_G(z)$. Note we sum over all possible $x_v \in [q]$ for $v \in V_i \setminus L_i$, but let the values x_v for $v \in L_i$ be undetermined and store the coefficient in the obtained polynomial of a certain monomial $\prod_{u\in R_i} x_u^{e_u}$. Indeed, it is easily seen that $P_G(z)$ equals $T^z_\ell[\emptyset, \emptyset]$, where $\emptyset$ is the unique 0-dimensional vector. By combining the appropriate recurrence for all values $T^z_i[d, e]$ with dynamic programming, the following lemma is proved in the full version [28].

▶ **Lemma 17** (★). *All values $T^z_i[d, e]$ can be computed in time $\mathrm{poly}(n) \cdot \sum_{i=1}^{\ell} T_i$, where*

$$T_i = |\mathcal{P}(l^i)| \cdot |\mathcal{P}(r^i)| = \prod_{v\in B_i} (\min\{d_{E_i}(v), d(v) - d_{E_i}(v)\} + 1).$$

Thus $P_G(z)$ can be computed in the time stated in Theorem 13. As discussed, $P_G(z) = 0$ if G has no proper q-coloring. Otherwise, ω isolates the set of proper q-colorings of G with probability at least $1/2$. Conditioned on this event we have $P_G(z) \neq 0$, where z is the weight of the unique minimum-weight q-coloring. Therefore we output YES if $P_G(z) \neq 0$ for some z and obtain the claimed probabilistic guarantee. This concludes the proof of Theorem 13.

As special cases of Theorem 13 we obtain Theorems 2 and 3.

Proof of Theorem 2. Given a linear layout $v_1, \dots, v_n$ of cutwidth k, define a nice path decomposition in which vertices are introduced in the order of the layout. After v_i is introduced, its incident edges to v_j with $j < i$ are introduced in arbitrary order. Forget v_i directly after the series of edge introductions that introduced its last incident edge.

As $v_1, \ldots, v_n$ has cutwidth at most k, for any bag B_i of this path decomposition the number of edges between V_i and $V \setminus V_i$ is at most k. Together with the edges incident on the most-recently introduced vertex v_j, these k edges are the only edges incident on B_i that are not in E_i. Consider the term $\prod_{v \in B_i}(\min\{d_{E_i}(v), d(v) - d_{E_i}(v)\} + 1)$. Vertex v_j contributes at most one factor n. For the remaining vertices in B_i, the only incident edges not in E_i are those in the cut of size at most k. By the AM-GM inequality, their contribution to the product is maximized when they are all incident to distinct vertices, in which case the algorithm of Theorem 13 runs in time $\mathcal{O}^*(2^k)$. ◀

Proof of Theorem 3. Follows from Theorem 13: $\min\{d_{E_i}(v), d(v) - d_{E_i}(v)\} \leq \lfloor d(v)/2 \rfloor$. ◀

4 Lower Bounds for Graph Coloring

In this section we discuss the main ideas behind our lower bounds, whose proofs are deferred to the full version [28]. We first start with Theorem 4, which rules out algorithms for solving 3-Coloring in time $\mathcal{O}^*((2-\varepsilon)^{\text{ctw}})$, even on *planar graphs*. (We remark that a companion paper [22] was the first to present lower bounds for planar graphs of bounded cutwidth.) The overall approach is based on the framework by Lokshtanov et al. [34]. We prove that an n-variable instance of CNF-SAT can be transformed in polynomial time into an equivalent instance of 3-Coloring on a planar graph G with a linear layout of cutwidth $n + \mathcal{O}(1)$. Consequently, saving ε in the base of the exponent when solving graph coloring would violate SETH. By employing clause-checking gadgets in the form of a path [27], crossover gadgets [21], and a carefully constructed ordering of the graph, we get the desired reduction.

The second lower bound, Theorem 5, rules out algorithms with running time $\mathcal{O}^*((\lfloor d/2 \rfloor + 1 - \varepsilon)^{\text{pw}})$ for solving q-Coloring for $q := \lfloor d/2 \rfloor + 1$ on graphs of maximum degree d and pathwidth pw, for any odd integer $d \geq 5$. The reduction employs *chains of cliques* to propagate assignments throughout a bounded-pathwidth graph. A t-chain of q-cliques is the graph obtained from a sequence of t vertex-disjoint q-cliques by selecting a distinguished *terminal* vertex in each clique and connecting it to the $(q-1)$ non-terminals in the previous clique. Any proper q-coloring of a chain assigns all terminals the same color, and terminals have $2(q-1)$ neighbors in the chain. Therefore, we can propagate a choice with q possibilities throughout a path decomposition. We encode truth assignments to variables of a CNF-SAT instance through colors given to the terminals of such chains. We enforce that the encoded truth assignment satisfies a clause, by enforcing that an assignment that does *not* satisfy the clause, is not the one encoded by the coloring. To check this, we take one terminal from each chain and connect it to a partner on a path gadget that forbids a specific coloring. Hence each vertex on a chain will receive at most one more neighbor, giving a maximum degree of $d := 2(q-1) + 1 = 2q - 1$ to represent a q-Coloring instance. Then solving this q-Coloring instance in $\mathcal{O}^*((\lfloor d/2 \rfloor + 1 - \varepsilon)^{\text{pw}}) = \mathcal{O}^*(((q-1) + 1 - \varepsilon)^{\text{pw}})$ time will contradict SETH for the same reason as in the earlier construction [34] showing the impossibility of $\mathcal{O}^*((q-\varepsilon)^{\text{pw}})$-time algorithms.

5 Conclusion

We showed how graph decompositions using small edge separators can be used to solve q-Coloring. The exponential parts of the running times of our algorithms are independent of q, which is a significant difference compared to algorithms for parameterizations based on vertex separators. The deterministic $\mathcal{O}^*(2^{\omega \cdot \text{ctw}})$ algorithm of Theorem 1 for the cutwidth parameterization follows cleanly from the bound on the rank of the partial solutions matrix.

It may serve as an insightful new illustration of the rank-based approach for dynamic-programming algorithms in the spirit of [8, 11, 12, 18].

One of the main take-away messages from this work from a practical viewpoint is the following. Suppose H is a subgraph of G connected to the remainder of the graph by k edges. Then any set of partial colorings $\mathcal{S}$ of H can be reduced to a subset $\mathcal{S}'$ of size 2^k, with the guarantee that if some coloring in $\mathcal{S}$ could be extended to a proper coloring of G, then this still holds for $\mathcal{S}'$. The reduction can be achieved by an application of Gaussian elimination, which has experimentally been shown to work well for speeding up dynamic programming for other problems [15]. We therefore believe the table-reduction steps presented here may also be useful when solving graph coloring over tree- or path decompositions, and can be applied whenever processing a separator consisting of few edges.

References

1 Noga Alon and Michael Tarsi. Colorings and orientations of graphs. *Combinatorica*, 12(2):125–134, 1992. `doi:10.1007/BF01204715`.

2 Noga Alon and Michael Tarsi. A note on graph colorings and graph polynomials. *J. Comb. Theory, Ser. B*, 70(1):197–201, 1997. `doi:10.1006/jctb.1997.1753`.

3 Nikhil Bansal, Marek Eliáš, Grigorios Koumoutsos, and Jesper Nederlof. Competitive algorithms for generalized k-server in uniform metrics. In *Proc. 29th SODA*, pages 992–1001, 2018. `doi:10.1137/1.9781611975031.64`.

4 Andreas Björklund. Coloring graphs having few colorings over path decompositions. In *Proc. 15th SWAT*, volume 53 of *LIPIcs*, pages 13:1–13:9, 2016. `doi:10.4230/LIPIcs.SWAT.2016.13`.

5 Andreas Björklund and Thore Husfeldt. Exact graph coloring using inclusion-exclusion. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*. Springer, 2008. `doi:10.1007/978-0-387-30162-4_134`.

6 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009. `doi:10.1137/070683933`.

7 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998. `doi:10.1016/S0304-3975(97)00228-4`.

8 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. `doi:10.1016/j.ic.2014.12.008`.

9 Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, 2008. `doi:10.1093/comjnl/bxm037`.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. `doi:10.1007/978-3-319-21275-3`.

11 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. In *Proc. 45th STOC*, pages 301–310. ACM, 2013. `doi:10.1145/2488608.2488646`.

12 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proc. 52nd FOCS*, pages 150–159, 2011. `doi:10.1109/FOCS.2011.23`.

13 J. A. de Loera. Gröbner bases and graph colorings. *Contributions to Algebra and Geometry*, 35(1):89–96, 1995.

14 Josep Díaz, Jordi Petit, and Maria J. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002. `doi:10.1145/568522.568523`.

15 Stefan Fafianie, Hans L. Bodlaender, and Jesper Nederlof. Speeding up dynamic programming with representative sets: An experimental evaluation of algorithms for Steiner tree on tree decompositions. *Algorithmica*, 71(3):636–660, 2015. doi:10.1007/s00453-014-9934-0.

16 Michael R. Fellows, Bart M. P. Jansen, and Frances Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *European J. Combin.*, 34(3):541–566, 2013. doi:10.1016/j.ejc.2012.04.008.

17 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010. doi:10.1137/080742270.

18 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016. doi:10.1145/2886094.

19 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In *Proc. 8th IPEC*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013. doi:10.1007/978-3-319-03898-8_15.

20 Robert Ganian. Twin-cover: Beyond vertex cover in parameterized algorithmics. In Dániel Marx and Peter Rossmanith, editors, *Proc. 6th IPEC*, volume 7112 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2011. doi:10.1007/978-3-642-28050-4_21.

21 M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.

22 Bas A.M. van Geffen, Bart M.P. Jansen, Arnoud A.W.M. de Kroon, and Rolf Morel. Lower bounds for dynamic programming on planar graphs of bounded cutwidth. *CoRR*, 2018. arXiv:1806.10513.

23 Archontia C. Giannopoulou, Michal Pilipczuk, Jean-Florent Raymond, Dimitrios M. Thilikos, and Marcin Wrochna. Cutwidth: Obstructions and algorithmic aspects. In *Proc. 11th IPEC*, volume 63 of *LIPIcs*, pages 15:1–15:13, 2016. doi:10.4230/LIPIcs.IPEC.2016.15.

24 Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Cliquewidth III: The odd case of graph coloring parameterized by cliquewidth. In *Proc. 29th SODA*, pages 262–273, 2018. doi:10.1137/1.9781611975031.19.

25 Russel Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

26 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

27 Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. In *Proc. 10th CIAC*, Lecture Notes in Computer Science, pages 345–356, 2017. doi:10.1007/978-3-319-57586-5_29.

28 Bart M. P. Jansen and Jesper Nederlof. Computing the chromatic number using graph decompositions via matrix rank. *CoRR*, 2018. arXiv:1806.10501.

29 T.R. Jensen and B. Toft. *Graph Coloring Problems*. Wiley interscience publication. Wiley, 1995.

30 David S. Johnson, Anuj Mehrotra, and Michael A. Trick. Special issue on computational methods for graph coloring and its generalizations. *Discrete Applied Mathematics*, 156(2):145–146, 2008. doi:10.1016/j.dam.2007.10.007.

31 Daniel Kobler and Udi Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Applied Mathematics*, 126(2-3):197–221, 2003. doi:10.1016/S0166-218X(02)00198-1.

32 Ephraim Korach and Nir Solel. Tree-width, path-width, and cutwidth. *Discrete Applied Mathematics*, 43(1):97–101, 1993. doi:10.1016/0166-218X(93)90171-J.

33 R.M. R. Lewis. *A Guide to Graph Colouring: Algorithms and Applications*. Springer Publishing Company, 2015.

34 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. In *Proc. 22nd SODA*, pages 777–789, 2011. doi:10.1137/1.9781611973082.61.

35 L. Lovász. Bounding the independence number of a graph. In Achim Bachem, Martin Grötschel, and Bemhard Korte, editors, *Bonn Workshop on Combinatorial Optimization*, volume 66, pages 213–223. North-Holland, 1982. doi:10.1016/S0304-0208(08)72453-8.

36 László Lovász. Flats in matroids and geometric graphs. In *Combinatorial surveys (Proc. Sixth British Combinatorial Conf.)*, pages 45–86. Academic Press London, 1977.

37 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987. doi:10.1007/BF02579206.

38 P.M. Pardalos, T. Mavridou, and J. Xue. *The graph coloring problem: A bibliographic survey*, volume 2, pages 331–395. Kluwer Academic Publishers, Boston, 1998.

39 Sigve Hortemo Sæther and Jan Arne Telle. Between treewidth and clique-width. *Algorithmica*, 75(1):218–253, 2016. doi:10.1007/s00453-015-0033-7.

40 Jan Arne Telle and Andrzej Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Discrete Math.*, 10(4):529–550, 1997. doi:10.1137/S0895480194275825.

41 Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Cutwidth I: A linear time fixed parameter algorithm. *J. Algorithms*, 56(1):1–24, 2005. doi:10.1016/j.jalgor.2004.12.001.

42 Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Cutwidth II: algorithms for partial w-trees of bounded degree. *J. Algorithms*, 56(1):25–49, 2005. doi:10.1016/j.jalgor.2004.12.003.

Polynomial Kernels for Hitting Forbidden Minors under Structural Parameterizations

Bart M. P. Jansen
Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
b.m.p.jansen@tue.nl
https://orcid.org/0000-0001-8204-1268

Astrid Pieterse
Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
a.pieterse@tue.nl
https://orcid.org/0000-0003-3721-6721

Abstract

We investigate polynomial-time preprocessing for the problem of hitting forbidden minors in a graph, using the framework of kernelization. For a fixed finite set of graphs $\mathcal{F}$, the $\mathcal{F}$-Deletion problem is the following: given a graph G and integer k, is it possible to delete k vertices from G to ensure the resulting graph does not contain any graph from $\mathcal{F}$ as a minor? Earlier work by Fomin, Lokshtanov, Misra, and Saurabh [FOCS'12] showed that when $\mathcal{F}$ contains a planar graph, an instance (G, k) can be reduced in polynomial time to an equivalent one of size $k^{\mathcal{O}(1)}$. In this work we focus on structural measures of the complexity of an instance, with the aim of giving nontrivial preprocessing guarantees for instances whose solutions are large. Motivated by several impossibility results, we parameterize the $\mathcal{F}$-Deletion problem by the size of a vertex modulator whose removal results in a graph of constant treedepth η.

We prove that for each set $\mathcal{F}$ of connected graphs and constant η, the $\mathcal{F}$-Deletion problem parameterized by the size of a treedepth-η modulator has a polynomial kernel. Our kernelization is fully explicit and does not depend on protrusion reduction or well-quasi-ordering, which are sources of algorithmic non-constructivity in earlier works on $\mathcal{F}$-Deletion. Our main technical contribution is to analyze how models of a forbidden minor in a graph G with modulator X, interact with the various connected components of $G - X$. Using the language of labeled minors, we analyze the fragments of potential forbidden minor models that can remain after removing an optimal $\mathcal{F}$-Deletion solution from a single connected component of $G - X$. By bounding the number of different types of behavior that can occur by a polynomial in $|X|$, we obtain a polynomial kernel using a recursive preprocessing strategy. Our results extend earlier work for specific instances of $\mathcal{F}$-Deletion such as Vertex Cover and Feedback Vertex Set. It also generalizes earlier preprocessing results for $\mathcal{F}$-Deletion parameterized by a vertex cover, which is a treedepth-one modulator.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis, Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Kernelization, $\mathcal{F}$-minor free deletion, Treedepth modulator, Structural parameterization

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.48

Related Version A full version is available at [26], `https://arxiv.org/abs/1804.08885`.

Funding This work was supported by NWO Veni grant "Frontiers in Parameterized Preprocessing" and NWO Gravitation grant "Networks".

© Bart M. P. Jansen and Astrid Pieterse;
licensed under Creative Commons License CC-BY

1 Introduction

How, and under which circumstances, can a polynomial-time algorithm prune the easy parts of an NP-hard problem input, without changing its answer? This question can rigorously be answered using the notion of kernelization [1, 23, 29] which originated in parameterized complexity theory [8, 12] where it can be naturally framed. After choosing a *complexity parameter* for the NP-hard problem of interest, which associates to every input $x \in \Sigma^*$ an integer $k \in \mathbb{N}$ that expresses its difficulty under the chosen type of measurement, the theory postulates that a good preprocessing algorithm can be captured by the notion of a *polynomial kernelization*: a polynomial-time algorithm that, given a parameterized instance $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs an instance (x', k') with the same answer whose size is bounded polynomially in k. Not all parameterized problems admit polynomial kernelizations, and one can find meaningful ways to preprocess an NP-hard problem by studying those parameterizations for which it does. The study of kernelization has blossomed over the last decade, resulting in a myriad of interesting techniques for obtaining polynomial kernelizations [3, 15, 24, 31, 34], as well as frameworks for proving the non-existence of polynomial kernelizations under complexity-theoretic assumptions [1, 2, 11, 13, 20].

Originally, the study of kernelization focused on the *natural parameterizations* of (the decision variants of) search problems, where the complexity parameter k measures the size of the solution. A classic example [7, 35] is that an instance (G, k) of the k-Vertex Cover problem, which asks whether an undirected graph G has a vertex cover of size k, can efficiently be reduced to an equivalent instance with at most $2k$ vertices. This guarantees that efficient pruning can be done on large inputs that have small vertex covers. However, such guarantees are meaningless when the smallest vertex cover contains more than half the vertices. By choosing a parameter that measures the structure of the input graph, rather than the size of the desired solution, one can hope to develop provably good preprocessing procedures even for inputs whose solutions are large. An early example of this approach was given by Jansen and Bodlaender [25], who showed that an instance of the Vertex Cover problem can efficiently be reduced to size $\mathcal{O}(\ell^3)$, where ℓ is the size of a smallest *feedback vertex set* in G: Vertex Cover parameterized by the size of a feedback vertex set has a cubic-vertex kernel. The result effectively conveys that large instances of Vertex Cover that are ℓ vertex-deletions away from being acyclic, can be shrunk to size $\mathcal{O}(\ell^3)$ in polynomial time.

Problem statement. To understand the power of polynomial-time preprocessing algorithms over inputs to NP-hard problems that exhibit some structural regularities, but whose solutions are generally large, we set out to answer the following question:

For which structural parameterizations of NP-hard graph problems is it possible to obtain polynomial kernelizations?

Our goal is to answer this question for a rich class of problems, in terms of a rich class of structural parameterizations. Existing lower bounds show that, in general graphs, it is unlikely that a logical characterization exists of the problems admitting polynomial kernelizations for structural parameterizations (cf. [16, §1]), even though meta-theorems in terms of logical definability or finite integer index are possible when dealing with inputs from sparse graph families [3, 21]. We therefore target the class of $\mathcal{F}$-Minor-Free Deletion problems, henceforth abbreviated as $\mathcal{F}$-Deletion problems, to capture a wide class of NP-hard graph problems. Such a problem is instantiated by specifying a finite set $\mathcal{F}$ of forbidden minors. An input then consists of a graph G and integer k, and asks whether it is possible to find a set $Y \subseteq V(G)$ of size k such that $G - Y$ contains no graph from $\mathcal{F}$ as a minor. This is a rich class of problems: by choosing $\mathcal{F} = \{K_2\}$ we obtain Vertex Cover, for $\mathcal{F} = \{K_3\}$ we have Feedback Vertex Set, and for $\mathcal{F} = \{K_5, K_{3,3}\}$ we obtain the problem of making a graph planar by vertex deletions. The kernelization complexity of the solution-size parameterization of $\mathcal{F}$-Deletion has been the subject of intensive research [17, 18, 22, 28, 40]. In this work we attempt to find the widest class of structural parameterizations for which $\mathcal{F}$-Deletion admits polynomial kernels, continuing a long line of investigation into structural parameterizations for Vertex Cover [4, 19, 25, 30, 31, 33], Feedback Vertex Set [27, 32], and other $\mathcal{F}$-Deletion problems [16, 21].

When it comes to measuring graph complexity, a natural choice is to consider a *width measure* such as treewidth. Alas, it has long been known that even Vertex Cover, the simplest $\mathcal{F}$-Deletion problem, does not admit a polynomial kernelization when parameterized by the treewidth of the input graph, assuming $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$.[1] Generally speaking, graph problems do not admit polynomial kernels under parameterizations that attain the maximum, rather than the sum, of the values of the connected components. We therefore use the *vertex-deletion* distance to simple graph classes $\mathcal{G}$ as the parameter. The aforementioned result by Jansen and Bodlaender [25] shows that Vertex Cover has a polynomial kernelization when parameterized by the vertex-deletion distance to an acyclic graph, i.e., to a graph of treewidth one. Unfortunately this formulation leaves little room for generalizations: no polynomial kernelization is possible parameterized by the distance to a graph of treewidth two [10, Theorem 11], or even pathwidth two.[2] We therefore cannot use the deletion distance to constant treewidth (TW) or pathwidth (PW) as our graph parameter, and use the deletion distance to constant *treedepth* (TD) instead. The parameter treedepth has recently attracted much interest [6, 14, 38], sometimes allowing better upper bounds than are possible in terms of treewidth [21, 37]. It plays an important role in the study of structural sparsity [36]. All graphs G satisfy $\textsc{td}(G) \geq \textsc{pw}(G) \geq \textsc{tw}(G)$, so graphs of constant treedepth are more restricted than those of constant treewidth. We therefore study the following problem for a fixed set $\mathcal{F}$ of connected graphs and constant $\eta \geq 1$.

$\mathcal{F}$-Deletion parameterized by treedepth-η modulator **Parameter:** $|X|$.
Input: A graph G, integer k, and a modulator $X \subseteq V(G)$ such that $\textsc{td}(G - X) \leq \eta$.
Question: Is there a set $Y \subseteq V(G)$ of size k such that $G - Y$ is $\mathcal{F}$-minor-free?

The restriction that $\mathcal{F}$ contains only connected graphs is needed to ensure that a solution on a disconnected graph can be formed from solutions on its connected components, which we require in some of our proofs. The same assumption was used by Fomin et al. [18] to build a single-exponential FPT algorithm when $\mathcal{F}$ contains a planar graph, and was later lifted in follow-up work by Kim et al. [28].

For technical reasons, we assume that a modulator X is given in the input. If no modulator is known, one can compute an approximate modulator and use it as X. For example, Gajarský et al. [21, Lemma 4.2] showed that a modulator of size at most 2^η times the optimum can be found in quadratic time. Our problem setting is related to that of Gajarský et al. [21]. They studied kernelization for a general class of graph problems that

[1] Bodlaender et al. [2, Theorem 1] show a superpolynomial kernelization lower bound for Independent Set parameterized by treewidth. Since the parameter is not related to the solution size, this is equivalent to Vertex Cover parameterized by treewidth. The lower bound holds under the assumption that $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$, which we implicitly assume when stating further lower bounds in this section.
[2] The lower bound is stated for distance to treewidth two, but the same proof works for pathwidth two.

includes $\mathcal{F}$-Deletion, parameterized by a constant-treedepth modulator, but under the additional restriction that the input graph has bounded expansion or is nowhere dense. Under this severe restriction they obtained kernelizations of linear size for a wide range of problems. This prompted Somnath Sikdar during the 2013 Workshop on Kernelization [9] to ask which types of problems admit polynomial kernelizations in *general graphs*, when parameterized by a constant-treedepth modulator; we address this question in this work.

Our results. Our main result proves the existence of polynomial kernelizations for $\mathcal{F}$-Deletion parameterized by a modulator whose removal leaves a graph of constant treedepth.

▶ **Theorem 1.** *For every fixed finite set $\mathcal{F}$ of connected graphs and every constant η, the $\mathcal{F}$-*Deletion *problem parameterized by a treedepth-η modulator has a polynomial kernelization.*

This answers a question posed by Bougeret and Sau [4] (cf. [5]). They obtained polynomial kernels for Vertex Cover parameterized by a constant-treedepth modulator, and asked whether their result can be extended to the Feedback Vertex Set problem. As Feedback Vertex Set is an $\mathcal{F}$-Deletion problem for $\mathcal{F} = \{K_3\}$, Theorem 1 shows that this is indeed the case. Theorem 1 greatly generalizes an earlier result of Fomin, Jansen, and Pilipczuk [16, Corollary 1], who proved that $\mathcal{F}$-Deletion parameterized by a *vertex cover* has a polynomial kernel for every fixed $\mathcal{F}$; note that a vertex cover is precisely a treedepth-1 modulator.

Our kernelization is fully explicit and does not depend on protrusion replacement techniques or well-quasi-ordering, which are sources of algorithmic non-constructivity in other works [17, 18] on kernelization for $\mathcal{F}$-Deletion. Moreover, our general theorem allows $\mathcal{F}$ to be any set of connected graphs, including nonplanar ones. In contrast, the kernelization for the solution-size parameterization by Fomin et al. [18] only applies when $\mathcal{F}$ contains at least one planar graph. Hence they only capture problems where, after removing a solution, the remaining graph has constant treewidth [39]. In our case, even though the parameter value is expressed in terms of a modulator to a graph of constant treedepth and therefore constant treewidth, the graphs that result after removing an optimal solution may have unbounded treewidth. This occurs, for example, when using $\mathcal{F} = \{K_5, K_{3,3}\}$ to capture the Vertex Planarization problem. (Whether the solution-size parameterization of Vertex Planarization has a polynomial kernel is a notorious open problem [18].)

The degree of the polynomial in the kernel size bound grows very quickly with η. We prove that this is unavoidable, even for the simplest case of Vertex Cover.

▶ **Theorem 2 (★).** *For every $\eta \geq 6$, the* Vertex Cover *problem parameterized by the size of a given treedepth-η modulator X does not admit a kernelization of bitsize $\mathcal{O}(|X|^{2^{\eta-4}-\varepsilon})$ for any $\varepsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.*

Techniques. To obtain a polynomial kernel for an instance (G, X, k) of $\mathcal{F}$-Deletion, the main challenge is to understand how the connected components $\mathcal{C}$ of $G - X$ interact through their connections to the modulator X. Using the language of labeled minors, we analyze how minor models of a forbidden graph in $\mathcal{F}$ may intersect the various components of $G - X$. Using these insights, we are able to characterize which components of $\mathcal{C}$ affect the structure of optimal solutions in an essential way. On a high level, the kernelization strategy is as follows. We use the fact that a single constant-treedepth component can be analyzed efficiently, to identify a subset $\mathcal{C}'$ of $\mathcal{C}$ that contains $|X|^{\mathcal{O}(1)}$ essential components under our characterization. We prove that the remaining ones can be safely removed, because their interaction with the rest of the instance can be ignored. Formally speaking, we show that any optimal solution on $G' := G[X \cup \bigcup_{C \in \mathcal{C}'} C]$ can be lifted to a solution on G by including $\Delta = \sum_{C \in \mathcal{C} \setminus \mathcal{C}'} \textsc{opt}_{\mathcal{F}}(C)$ additional vertices: (G, X, k) is a YES-instance if and only if $(G', X, k - \Delta)$ is. This effectively shows that there is an optimal solution Y on G in which the non-essential components act in isolation: Y does not delete more vertices from such a component C, than would be deleted by a solution on the graph $G[C]$.

The overall kernelization follows straight-forwardly from this pruning of non-essential components by a recursive approach, similarly as in earlier work [4, 21]. The main challenge is therefore to understand which components are essential and which are not, and this is where our contribution lies. We present a stand-alone combinatorial lemma that captures our key insight in this direction. To state it, we introduce some terminology.

We work with a nonstandard notion of labeled graphs. For a finite set X, an *X-labeled graph* is a graph in which each vertex is assigned a (possibly empty) subset of X as its labelset; we stress that multiple vertices may carry the same label on their labelset. The minor relation on graphs extends to labeled graphs in a natural way: a labeled graph H is a minor of a labeled graph G, if H can be obtained from G by repeatedly deleting an edge, deleting a vertex, deleting a label from the labelset of a vertex, or contracting an edge. When contracting an edge $\{u, v\}$ into a single vertex w, the labelset of w is formed as the union of the labelsets of u and v.

For a collection $\mathcal{S}$ of vertex subsets of an X-labeled graph C, and a set of X-labeled graphs $\mathcal{Q}$, we say that all $Y \in \mathcal{S}$ *leave a $\mathcal{Q}$-minor* in C, if for all $Y \in \mathcal{S}$ the graph $C - Y$ contains some graph $H \in \mathcal{Q}$ as a labeled minor. We say that a set $\mathcal{Q}$ of X-labeled graphs is θ-saturated for an integer θ, if for each subset $X' \subseteq X$ of size θ, the graph consisting of one vertex with labelset X' belongs to $\mathcal{Q}$. Our main lemma states that if all optimal solutions to $\mathcal{F}$-DELETION on C leave a $\mathcal{Q}$-minor for some suitably saturated $\mathcal{Q}$, then there is a small subset $\mathcal{Q}^*$ for which the same holds.

▶ **Lemma 3** (Main lemma ★). *Let $\mathcal{F}$ be a finite set of (unlabeled) connected graphs, let X be a set of labels, let $\mathcal{Q}$ be a $(\min_{H \in \mathcal{F}} |V(H)|)$-saturated set of connected X-labeled graphs of at most $\max_{H \in \mathcal{F}} |E(H)| + 1$ vertices each, and let C be an X-labeled graph. If all optimal solutions to $\mathcal{F}$-DELETION on C leave a $\mathcal{Q}$-minor, then there is a subset $\mathcal{Q}^* \subseteq \mathcal{Q}$ whose size depends only on $(\mathcal{F}, \textsc{td}(C))$, such that all optimal solutions leave a $\mathcal{Q}^*$-minor.*

In several aspects, the statement in the lemma is best-possible. In particular, we will show in Section 3 that the dependence of the size of $\mathcal{Q}^*$ on $\textsc{td}(G)$ rather than $\textsc{tw}(G)$ is essential and that the precondition that $\mathcal{Q}$ is $\mathcal{O}(1)$-saturated cannot be avoided.

Lemma 3 is the cornerstone in our understanding of which components of $G - X$ are essential. In our applications of the lemma, the graph C consists of a connected component of $G - X$ whose labels encode the adjacency of those vertices to the modulator X. The set $\mathcal{Q}$ contains potential fragments of models of forbidden $\mathcal{F}$-minors, again labeled by adjacency to X, which we may be interested in destroying in C so that connections through X cannot form $\mathcal{F}$-minors with fragments that remain in other components of $G - X$. The lemma then essentially says that if it is not possible to select a solution that deletes a minimum number of vertices from C while simultaneously destroying all fragments in $\mathcal{Q}$, then there is a bounded-size subset of fragments $\mathcal{Q}^*$ that cannot all be destroyed by such a solution. The full importance of Lemma 3 will become clear in Section 4.

Organization. Section 2 provides basic preliminaries. In Section 3, we give some of the main ideas of the proof of Lemma 3. In Section 4 we show how Theorem 1 follows from a procedure that identifies relevant components. We give the procedure and its correctness proof later in the same section, while relying on Lemma 3.

The proof of Lemma 3 is very technical and requires us to develop a framework for analyzing minor models in boundaried labeled graphs. This proof, together with the proofs of other statements marked (★), can be found in the full version [26].

2 Preliminaries

For a positive integer n we use $[n]$ as a shorthand for $\{1, \ldots, n\}$. For a set S, let 2^S denote the set of all subsets of S. All graphs we consider are finite, undirected, and simple. A graph G consists of a vertex set $V(G)$ and edge set $E(G) \subseteq \binom{V(G)}{2}$. The open neighborhood of a vertex v is denoted $N_G(v)$. For a vertex set $S \subseteq V(G)$, its open neighborhood is $N_G(S) := \bigcup_{v \in S} N_G(v) \setminus S$. For an edge $\{u, v\}$ in a graph G, *contracting* $\{u, v\}$ results in the graph G' obtained from G by removing u and v, and replacing them by a new vertex w with $N_{G'}(w) = N_G(\{u, v\})$. For a vertex set $S \subseteq V(G)$, we use $G - S$ to denote the graph obtained from G by deleting all vertices in S and their incident edges. The subgraph of G induced by vertex set S is denoted $G[S]$.

▶ **Definition 4** (treedepth)**.** Treedepth is defined as follows. The trivial one-vertex graph has treedepth 1. The treedepth of a disconnected graph G with connected components $C^1, \ldots, C^t$ is $\max_{i \in [t]} \textsc{td}(C^i)$. The treedepth of a connected graph G is $\min_{v \in V(G)} \textsc{td}(G - \{v\}) + 1$.

▶ **Definition 5** (labeled graph)**.** Let X be a set. An *X-labeled graph* G is a graph G together with label function $L_G \colon V(G) \to 2^X$, assigning a (potentially empty) subset of labels to each vertex in G. The labeled graph G is θ-restricted if each vertex has at most θ labels.

If an edge $\{u, v\}$ is contracted in a labeled graph G to obtain a new vertex w, then the labelset of w is defined as $L_G(u) \cup L_G(v)$.

▶ **Definition 6** (minor model)**.** A *minor model* of a graph H in a graph G is a mapping $\varphi \colon V(H) \to 2^{V(G)}$ assigning a *branch set* $\varphi(v) \subseteq V(G)$ to each vertex $v \in V(H)$, such that:
- $G[\varphi(v)]$ is nonempty and connected for all $v \in V(H)$,
- $\varphi(v) \cap \varphi(u) = \emptyset$ for all $u \neq v \in V(H)$, and
- if $\{u, v\} \in E(H)$, then there exist $u' \in \varphi(u)$ and $v' \in \varphi(v)$ such that $\{u', v'\} \in E(G)$.

The third condition implies that one can find an *edge mapping* $\psi \colon E(H) \to E(G)$ such that:
- For all $\{u, v\} \in E(H)$, edge $\psi(\{u, v\})$ has one endpoint in $\varphi(u)$ and the other in $\varphi(v)$.

We will often use the existence of this edge mapping in our proofs.

For $S \subseteq V(H)$ we define $\varphi(S) := \bigcup_{v \in S} \varphi(v)$, and we define $\varphi(V(H))$ as the *range* of the minor model. A minor model φ of H in G is called *minimal* if no minor model φ' exists with $\varphi'(V(H)) \subsetneq \varphi(V(H))$.

▶ **Definition 7** (labeled minor model)**.** A *labeled minor model* of an X-labeled graph H in an X-labeled graph G is a mapping φ as in Definition 6, that additionally satisfies the following: for all $v \in V(H)$ and $\ell \in L_H(v)$ there exists $v' \in \varphi(v)$ such that $\ell \in L_G(v')$.

If G contains a (labeled) minor model of H, then we say that G contains H as a (labeled) minor and denote this as $H \preceq_m G$. Observe that G contains H as a (labeled) minor if and only if H can be obtained from G by deleting edges and vertices (and potentially labels), and contracting edges.

▶ **Lemma 8** (★)**.** *Let G and H be unlabeled graphs, let $X \subseteq V(G)$, and let φ be a minimal minor model of H in G. Then $\varphi(V(H))$ intersects at most $|X| + |V(H)| + |E(H)|$ connected components of $G - X$.*

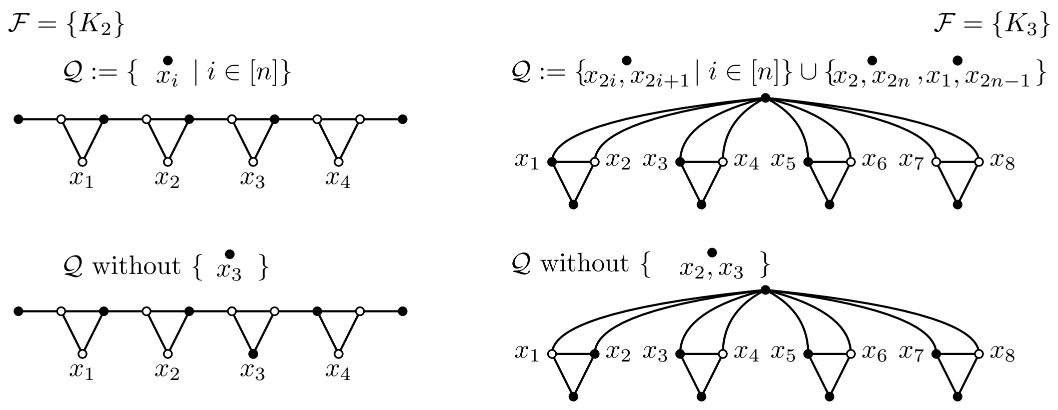

$\mathcal{F} = \{K_2\}$
$\mathcal{Q} := \{ \; x_i \mid i \in [n]\}$
x_1 x_2 x_3 x_4
$\mathcal{Q}$ without $\{ \; x_3 \; \}$
x_1 x_2 x_3 x_4
$\mathcal{F} = \{K_3\}$
$\mathcal{Q} := \{x_{2i}, x_{2i+1} \mid i \in [n]\} \cup \{x_2, x_{2n}, x_1, x_{2n-1}\}$
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8
$\mathcal{Q}$ without $\{ \; x_2, x_3 \; \}$
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8

Figure 1 Two constructions of graphs and sets $\mathcal{Q}$ for $n = 4$, where no optimal $\mathcal{F}$-deletion breaks $\mathcal{Q}$, but for any $Q \in \mathcal{Q}$ there exists an optimal $\mathcal{F}$-deletion breaking $\mathcal{Q} \setminus Q$. Top: any solution breaking both $\mathcal{F}$ and $\mathcal{Q}$ (white vertices at the top) is larger than $\textsc{opt}_{\mathcal{F}}$, but for any $Q \in \mathcal{Q}$ there is a solution of size $\textsc{opt}_{\mathcal{F}}$ breaking both $\mathcal{F}$ and $\mathcal{Q} \setminus \{Q\}$ (white vertices at the bottom).

We denote the size of an optimal $\mathcal{F}$-Deletion solution on G by $\textsc{opt}_{\mathcal{F}}(G)$, and the set of optimal solutions by $\textsc{optsol}_{\mathcal{F}}(G)$. In our bounds, we use the notation $\mathcal{O}_z(1)$ for some identifier(s) z to denote a constant that only depends on z.

▶ **Lemma 9** (★). *Let $\mathcal{F}$ be a fixed set of (unlabeled) graphs, let $\eta \geq 1$ be a constant, and let X be a set. For any set $\mathcal{Q}$ of X-labeled graphs and host graph C with $\textsc{td}(C) \leq \eta$, one can:*
- *compute $\textsc{opt}_{\mathcal{F}}(C)$ in $\mathcal{O}_{\mathcal{F},\eta}(|V(C)|)$ time;*
- *determine whether there is a solution $Y \in \textsc{optsol}_{\mathcal{F}}(C)$ such that $C - Y$ contains no graph from $\mathcal{Q}$ as a labeled minor, in time $f(L, \sum_{H \in \mathcal{Q}} |V(H)|, \eta) \cdot |V(C)|$ for some function f.*

Here L equals the number of elements of X that appear in the labelset of at least one vertex in at least one graph of $\mathcal{Q}$.

3 Overview of the main lemma

In this section we discuss Lemma 3, whose long and technical proof is deferred to the full version. The strength of the lemma comes from the fact that the bound on $|\mathcal{Q}^*|$ is *independent* of the size of the graph C and of the number of labels $|X|$ used on labelsets of vertices of C.

The statement of Lemma 3 is best-possible in several ways. First of all, the dependence of $|\mathcal{Q}^*|$ on $\textsc{td}(G)$ instead of $\textsc{tw}(G)$ is essential. In Figure 1 (left), a construction of a graph of treewidth 2 together with a set $\mathcal{Q}$ is shown. In this graph, no optimal $\{K_2\}$-deletion (Vertex Cover) breaks all graphs in $\mathcal{Q}$. However, for any $Q \in \mathcal{Q}$ there is an optimal vertex cover breaking $\mathcal{Q} \setminus \{Q\}$. The example in Figure 1 can easily be extended to arbitrary n, showing that there is a set $\mathcal{Q}$ with $|\mathcal{Q}| = n$ such that no optimal vertex cover breaks $\mathcal{Q}$, yet there is no $\mathcal{Q}^* \subsetneq \mathcal{Q}$ such that no optimal vertex cover breaks $\mathcal{Q}^*$. Since $|\mathcal{Q}|$ is not bounded in terms of $\textsc{tw}(G) = 2$ and $\mathcal{F} = \{K_2\}$, this shows that $\textsc{td}(G)$ cannot be replaced by $\textsc{tw}(G)$.

Secondly, the assumption that $\mathcal{Q}$ is $(\min_{H \in \mathcal{F}} |V(H)|)$-saturated cannot be avoided already for $\mathcal{F} = \{K_3\}$ (corresponding to Feedback Vertex Set). In Figure 1 (right) we show an example of a graph of treedepth 4 and a set $\mathcal{Q}$ of size $2n + 2$ that consist of single vertices of two labels each, where we again cannot properly bound the size of $\mathcal{Q}^*$. The example is shown for $n = 4$ but can easily be generalized to arbitrary n, without increasing the treedepth. For any $\mathcal{Q}^* \subsetneq \mathcal{Q}$ there exists an optimal $\mathcal{F}$-deletion breaking $\mathcal{Q}^*$, while $|\mathcal{Q}|$ is not bounded in terms of $\textsc{td}(G)$ and $\mathcal{F}$.

The proof of Lemma 3 follows an inductive strategy that mimics how a recursive algorithm would solve $\mathcal{F}$-Deletion on a bounded-treedepth graph C. We pick a vertex v whose removal decreases the treedepth, and branch on whether v is part of the solution or not. If so, we remove v and recurse on a graph of smaller treedepth; if not, then we continue looking for solutions in which v is forbidden to be removed. The process builds up a set S with the property that removing S decreases the treedepth by $|S|$, and we are only interested in solutions disjoint from S. This proceeds while $C - S$ remains connected; the branching depth is bounded since $|S| \leq \textsc{td}(C)$. When $C - S$ becomes disconnected, we must take a more involved approach. We recurse on each of the connected components of $C - S$ separately and find $\mathcal{F}$-Deletion solutions there. But solutions for different components of $C - S$ may not combine into a solution for C, since various fragments of $\mathcal{F}$-minors left behind in different components of $C - S$, may be combined through their connections to S to form a forbidden minor. For this reason, when we recurse on connected components of $C - S$ we place additional restrictions on the solutions chosen there, to ensure they also break *fragments* of $\mathcal{F}$-minors in such a way that the solutions can be properly combined.

Our approach to bound the size of $\mathcal{Q}^*$ is built on top of this inductive strategy. While branching over various ways to form an $\mathcal{F}$-Deletion solution, we additionally branch on what fragments of labeled $\mathcal{Q}$-minors are left behind by the solution in the various components of $C - S$. By exploiting the saturatedness of $\mathcal{Q}$ in a crucial way, we obtain the desired bound on $|\mathcal{Q}^*|$. The formalization of these ideas requires an extensive theory of how fragments of a forbidden minor in various components of $C - S$ may combine to form a forbidden minor in C, which is developed in Appendix B of the full version of the paper.

4 Kernelization for $\mathcal{F}$-Deletion

In this section we describe the recursive approach to kernelize the $\mathcal{F}$-Deletion problem using a constant-treedepth modulator. The correctness of this strategy will crucially depend on Lemma 3. Lemma 10 identifies essential components in the input.

▶ **Lemma 10.** *Let $\mathcal{F}$ be a finite set of connected graphs and let $\eta \geq 1$ be a constant. There is a polynomial-time algorithm that, given a graph G along with a modulator $X \subseteq V(G)$ such that $\textsc{td}(G - X) \leq \eta$, outputs an induced subgraph G' of G together with an integer Δ such that $\textsc{opt}_{\mathcal{F}}(G) = \textsc{opt}_{\mathcal{F}}(G') + \Delta$ and $G' - X$ has at most $|X|^{\mathcal{O}_{\mathcal{F},\eta}(1)}$ connected components.*

Before proving this lemma, we show how it implies Theorem 1.

▶ **Theorem 1.** *For every fixed finite set $\mathcal{F}$ of connected graphs and every constant η, the $\mathcal{F}$-Deletion problem parameterized by a treedepth-η modulator has a polynomial kernelization.*

Proof. Consider an input (G, X, k) to $\mathcal{F}$-Deletion. The proof is by induction on η.

$(\boldsymbol{\eta = 1})$ If $\textsc{td}(G - X) = 1$, then $G - X$ is an independent set and any connected component of $G - X$ contains one vertex. Apply Lemma 10 to find an induced subgraph G' of G and integer Δ such that $\textsc{opt}_{\mathcal{F}}(G) = \textsc{opt}_{\mathcal{F}}(G') + \Delta$, which implies that (G, X, k) has answer yes if and only if $(G', X, k - \Delta)$ has answer yes. Now $G' - X$ has $|X|^{\mathcal{O}_{\mathcal{F},1}(1)}$ single-vertex connected components. It follows that $G' - X$ has at most $|X| + |X|^{\mathcal{O}_{\mathcal{F},1}(1)}$ vertices, which is polynomial in $|X|$ for fixed $\mathcal{F}$. Hence $(G', X, k - \Delta)$ forms a polynomial kernel.

$(\boldsymbol{\eta > 1})$ For $\eta > 1$, we apply Lemma 10 on the input (G, X, k) and find G' and Δ as above. We will augment the modulator X into a superset X' to ensure that $\textsc{td}(G' - X') < \eta$. To this end, we consider each connected component C of $G' - X$. If C consists of a single vertex then its treedepth is already smaller than $\eta > 1$. Otherwise, C is a connected graph with more than one vertex, and by Definition 4 there is a vertex x_C such that $\textsc{td}(C - \{x_C\}) < \textsc{td}(C)$. Since the Treedepth problem parameterized by the target width is fixed-parameter tractable [38], and η is a constant, we can find such a vertex x_C by trying all options for x_C and computing the treewidth of the resulting graph in $f(\eta) \cdot n^{\mathcal{O}(1)}$ time. (Alternatively, we can compute a treedepth-decomposition of C using the algorithm of Reidl et al. [38] and take its root as x_C.) We initialize X' as X. For each component C of $G' - X$ with treedepth larger than one, we add the corresponding treedepth-decreasing vertex x_C to X'.

Since Lemma 10 guarantees that the number of connected components of $G' - X$ is polynomial in $|X|$ for fixed $\mathcal{F}$ and η, the resulting modulator X' has size polynomial in $|X|$. Moreover, it guarantees that $\textsc{td}(G' - X') < \eta$. Hence we now have an instance $(G', X', k - \Delta)$ of $\mathcal{F}$-Deletion parameterized by a treedepth-$(\eta - 1)$ modulator, with the same answer as (G, X, k). We apply the kernel for the parameterization by a treedepth-$(\eta - 1)$ modulator, which outputs an instance (G^*, X^*, k^*) with the same answer as $(G', X', k - \Delta)$ and therefore as (G, X, k). By induction, the size of G^* is bounded by some polynomial in $|X'|$, which in turn is bounded by a polynomial in $|X|$. Hence G^* has size $|X|^{\mathcal{O}_{\mathcal{F},\eta}(1)}$ for some suitably chosen constant, and we output (G^*, X^*, k^*) as the result of the kernelization. ◀

Now we prove Lemma 10.

Proof of Lemma 10. Let $\mathcal{C}$ be the connected components of $G - X$. To reduce their number, we have a single reduction rule stated in terms of labeled graphs. With each connected component $C \in \mathcal{C}$, we naturally associate an X-labeled graph C_L by assigning a vertex $v \in V(C)$ the labelset $N_G(v) \cap X$. We are interested in which of these labeled graphs have optimal $\mathcal{F}$-Deletion solutions that also hit certain fragments of potential $\mathcal{F}$-minor-models. We therefore define a set $\mathcal{H}$ which is a superset of the relevant fragments. We use $\|\mathcal{F}\|$ as a shorthand for $\max_{H \in \mathcal{F}} |V(H)|$. Let $\mathcal{H}$ consist of the connected $\|\mathcal{F}\|$-restricted X-labeled graphs that have at most $m_{\mathcal{F}} := \max_{H \in \mathcal{F}} |E(H)|$ edges. We consider two X-labeled graphs to be identical if there is an isomorphism between them that respects the labelsets.

▶ **Claim 11.** $|\mathcal{H}| \in |X|^{\mathcal{O}_{\mathcal{F}}(1)}$.

Proof. Graphs in $\mathcal{H}$ have at most $m_{\mathcal{F}} + 1$ vertices. There are less than $2^{(m_{\mathcal{F}}+1)^2}$ distinct choices for the graph structure of a member of $\mathcal{H}$, since there are less than 2^{n^2} different n-vertex graphs. For each vertex, there are less than $(|X|+1)^{\|\mathcal{F}\|}$ choices for a labelset of size at most $\|\mathcal{F}\|$. Hence each graph structure H can appear with less than $((|X|+1)^{\|\mathcal{F}\|})^{|V(H)|} \leq (|X|+1)^{\|\mathcal{F}\| \cdot (m_{\mathcal{F}}+1)}$ different choices of labeling function, giving an overall bound $|\mathcal{H}| \leq 2^{(m_{\mathcal{F}}+1)^2} \cdot (|X|+1)^{\|\mathcal{F}\| \cdot (m_{\mathcal{F}}+1)}$ that is polynomial in $|X|$. ⌟

Choose $\gamma \in \mathcal{O}_{\mathcal{F},\eta}(1)$ such that Lemma 3 guarantees that for this choice of $\mathcal{F}$ and the treedepth bound η, one can always find $\mathcal{Q}^* \subseteq \mathcal{Q}$ of size at most γ. Let $\rho := |X| + \max_{H \in \mathcal{F}}(|V(H)| + |E(H)|)$, and $\tau := |X| + 1 + \gamma \cdot \rho \in \mathcal{O}_{\mathcal{F},\eta}(|X|)$. Consider the following marking procedure.

▶ **Procedure 12.** *For each set $\mathcal{Q} \subseteq \mathcal{H}$ of size at most γ, do the following. Let*

$$\mathcal{C}_{\mathcal{Q}} := \{C \in \mathcal{C} \mid \forall Y \in \textsc{optsol}_{\mathcal{F}}(G[C]) \colon C_L - Y \text{ has a graph from } \mathcal{Q} \text{ as a labeled minor}\}.$$

Mark τ arbitrarily chosen components from $\mathcal{C}_{\mathcal{Q}}$, or mark all of them if there are fewer than τ.

Let $\mathcal{C}' \subseteq \mathcal{C}$ denote the marked components, $G' := G[X \cup \bigcup_{C \in \mathcal{C}'} C]$, and let $\Delta := \sum_{C \in \mathcal{C} \setminus \mathcal{C}'} \textsc{opt}_{\mathcal{F}}(G[C])$. The procedure can be executed in polynomial time, using variants of Courcelle's theorem to find the sets $\mathcal{C}_{\mathcal{Q}}$. We explain how this is done in Lemma 9. Since $\gamma \in \mathcal{O}_{\mathcal{F},\eta}(1)$, the number of subsets of $\mathcal{H}$ over which we iterate is polynomial in $|\mathcal{H}|$ and therefore in $|X|$. Since the graphs in $\mathcal{Q}$ are $\|\mathcal{F}\|$-restricted, the number of labels involved is constant for fixed $\mathcal{F}$ and η, and therefore Lemma 9 guarantees a polynomial running time.

▶ **Claim 13.** $|\mathcal{C}'| \leq |X|^{\mathcal{O}_{\mathcal{F},\eta}(1)}$.

Proof. The procedure loops over $|X|^{\mathcal{O}_{\mathcal{F},\eta}(1)}$ subsets $\mathcal{Q}$. For each such set, we mark at most $\tau = |X| + 1 + \gamma \cdot \rho \in \mathcal{O}_{\mathcal{F},\eta}(|X|)$ components. ⌟

The pair (G', Δ) is the desired outcome of Lemma 10. It remains to prove that $\textsc{opt}_{\mathcal{F}}(G) = \textsc{opt}_{\mathcal{F}}(G') + \Delta$. This follows from Claim 14 by induction.

▶ **Claim 14.** *For any unmarked component* $C^* \in \mathcal{C} \setminus \mathcal{C}'$: $\textsc{opt}_{\mathcal{F}}(G) = \textsc{opt}_{\mathcal{F}}(G - V(C^*)) + \textsc{opt}_{\mathcal{F}}(G[C^*])$.

Proof. Let $\widehat{G} := G - V(C^*)$. Clearly, any solution for the graph G can be partitioned into a solution for $\widehat{G}$ and a solution for $G[C^*]$, so that $\textsc{opt}_{\mathcal{F}}(G) \geq \textsc{opt}_{\mathcal{F}}(\widehat{G}) + \textsc{opt}_{\mathcal{F}}(G[C^*])$. We focus on proving the converse. Let $\widehat{Y} \in \textsc{optsol}_{\mathcal{F}}(\widehat{G})$ be an optimal solution on $\widehat{G}$. Let $X_0 := X \setminus \widehat{Y}$ and let $\mathcal{H}_0 \subseteq \mathcal{H}$ contain those graphs for which the labelset of each vertex is contained in X_0. Now define:

$$\begin{aligned}\mathcal{Q} := \{H \in \mathcal{H}_0 \mid {} & \text{there are fewer than } \rho \text{ components } C \text{ of } \widehat{G} - X \\ & \text{whose } X\text{-labeled version } C_L - \widehat{Y} \text{ contains } H \text{ as } X\text{-labeled minor}\}.\end{aligned} \tag{1}$$

Intuitively, one may think of $\mathcal{Q}$ as those labeled graphs (that represent potential fragments of forbidden $\mathcal{F}$-minors) that can be realized in only few ($\rho \in \mathcal{O}_{\mathcal{F}}(|X|)$) components of $\widehat{G} - X$ after removing the solution $\widehat{Y}$. When lifting the solution $\widehat{Y}$ in $\widehat{G}$ to a solution in G by adding a solution in C^*, it will be crucial to break all X-labeled minor models of $\mathcal{Q}$ in C^*; the fragments $\mathcal{H}_0 \setminus \mathcal{Q}$ that remain in *many* different components turn out to be irrelevant.

For a subset $X' \subseteq X_0$ of labels, let $I_{X'}$ be the labeled graph consisting of a single vertex with labelset X'. Let $n_{\mathcal{F}} := \min_{H \in \mathcal{F}} |V(H)|$ and observe that $n_{\mathcal{F}} \leq \rho$. We prove:

$$\forall X' \subseteq X_0, |X'| = n_{\mathcal{F}} \colon I_{X'} \in \mathcal{Q}. \tag{2}$$

Suppose $I_{X'} \notin \mathcal{Q}$ for suitable X'. Then there are $\rho \geq n_{\mathcal{F}}$ components of $\widehat{G} - X$ that have $I_{X'}$ as labeled minor after removing the solution $\widehat{Y}$. Take $n_{\mathcal{F}}$ such components $C_L^1, \ldots, C_L^{n_{\mathcal{F}}}$, and associate each one to a distinct vertex of $X' \subseteq V(\widehat{G}) \setminus \widehat{Y}$. The fact that $I_{X'}$ is a labeled minor of $C_L^i - \widehat{Y}$ for each i, implies that in each such component there is a connected vertex subset $S_i \subseteq V(C_L^i) \setminus \widehat{Y}$ such that each label of X' appears at least once on a vertex of S_i. Considering the corresponding vertex subset in $\widehat{G} - \widehat{Y}$ and taking into account that the labeling of C_L^i represents adjacency to X in G, this implies that we can contract each S_i into a single vertex s_i that becomes adjacent to all vertices of X'. Then contract each s_i into a distinct vertex of X': these minor operations on graph $\widehat{G} - \widehat{Y}$ turn X' into a clique of size $n_{\mathcal{F}}$. Hence any graph on $n_{\mathcal{F}}$ vertices is a minor of $\widehat{G} - \widehat{Y}$, contradicting that $\widehat{G} - \widehat{Y}$ is $\mathcal{F}$-minor-free since $\mathcal{F}$ has a graph on $n_{\mathcal{F}}$ vertices. So (2) holds.

Now consider the unmarked component C^* in the statement of Claim 14, and consider its labeled version C_L^*. We say that a vertex set Y *breaks* the minor models of the X_0-labeled

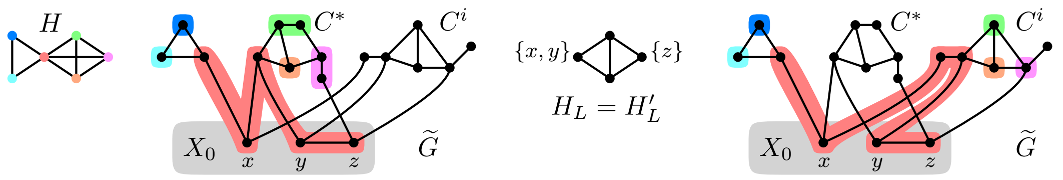


Figure 2 This figure shows how to define H_L based on H and $\widetilde{G}$, and how to modify the minor model of H in $\widetilde{G}$ such that it uses fewer vertices of C^*, in the proof of (4) in Claim 14.

graphs $\mathcal{Q}$ in C^*_L, or simply *breaks* $\mathcal{Q}$ in C^*_L, if $C^*_L - Y$ does not contain any graph in $\mathcal{Q}$ as a labeled minor. We first show the following.

$$\exists Y^* \in \text{OPTSOL}_{\mathcal{F}}(G[C^*])\colon\ Y^* \text{ breaks } \mathcal{Q} \text{ in } C^*_L. \tag{3}$$

To establish (3), assume that no solution of size $\text{OPT}_{\mathcal{F}}(G[C^*])$ in $G[C^*]$ breaks $\mathcal{Q}$. We will use Lemma 3, together with our marking scheme, to argue for a contradiction. Observe that (2) implies that $\mathcal{Q}$ is an $n_{\mathcal{F}}$-saturated set of X_0-labeled graphs. If no optimal solution on $G[C^*]$ breaks $\mathcal{Q}$, then by Lemma 3 there is a set $\mathcal{Q}^* \subseteq \mathcal{Q}$ of size at most γ such that no optimal solution on $G[C^*]$ breaks $\mathcal{Q}^*$. Since the assumption that (3) does not hold means that the unmarked C^* was eligible to be marked for the set $\mathcal{C}_{\mathcal{Q}^*}$ in our procedure above, it has marked τ other components $C^1, \ldots, C^\tau \in \mathcal{C}_{\mathcal{Q}^*}$ of $G - X$. For each $i \in [\tau]$, there is no $\mathcal{F}$-Deletion solution of size $\text{OPT}_{\mathcal{F}}(G[C^i])$ in $G[C^i]$ that breaks $\mathcal{Q}^*$ in the labeled version C^i_L. Since $\mathcal{Q}^* \subseteq \mathcal{Q}$, by (1) we have for each graph $H \in \mathcal{Q}^*$ that there are fewer than ρ components C^i among $C^1, \ldots, C^\tau$ for which $C^i_L - \widehat{Y}$ contains H as a labeled minor. Since $|\mathcal{Q}^*| \leq \gamma$, it follows that there are at most $\gamma \cdot \rho$ indices $i \in [\tau]$ for which $C^i_L - Y$ contains some graph from $\mathcal{Q}^*$ as a labeled minor. But since $\tau = |X| + 1 + \gamma \cdot \rho$, there are at least $|X|+1$ components C^i_L in which all $\mathcal{Q}^*$-minors are broken by $\widehat{Y}$. Since no *optimal* solution breaks $\mathcal{Q}^*$ in the marked components, we have $|\widehat{Y} \cap V(C^i)| > \text{OPT}_{\mathcal{F}}(G[C^i])$ for at least $|X|+1$ components. But this contradicts that $\widehat{Y}$ is an optimal solution to $\mathcal{F}$-Deletion on $\widehat{G}$: since $\mathcal{F}$ consists of connected graphs, we can form a solution $\widehat{Y}'$ by taking X together with a set of size $\text{OPT}_{\mathcal{F}}(\widehat{G}[C])$ from each component C of $\widehat{G} - X$. Since $|\widehat{Y}' \cap V(C)| \leq |\widehat{Y} \cap V(C)|$ for all $C \in \mathcal{C}$, with strict inequality for at least $|X| + 1$ components, we have $|\widehat{Y}'| < |\widehat{Y}|$. This contradicts that $\widehat{Y}$ is an optimal solution and establishes (3).

Hence there exists a solution Y^* in C^*_L breaking $\mathcal{Q}$ of size $\text{OPT}_{\mathcal{F}}(G[C^*])$. We prove:

$$\widehat{Y} \cup Y^* \text{ is a solution to } \mathcal{F}\text{-Deletion on } G. \tag{4}$$

This will complete the proof of Claim 14, since $|\widehat{Y} \cup Y^*| = \text{OPT}_{\mathcal{F}}(\widehat{G}) + \text{OPT}_{\mathcal{F}}(G[C^*])$. Assume for a contradiction that $\widetilde{G} := G - (\widehat{Y} \cup Y^*)$ contains some graph $H \in \mathcal{F}$ as a minor. Consider a minimal minor model of H in $\widetilde{G}$, which is given by a vertex mapping $\varphi\colon V(H) \to 2^{V(\widetilde{G})}$, and let $\psi\colon E(H) \to E(\widetilde{G})$ be a corresponding edge mapping.

Out of all possible minimal minor models of H in $\widetilde{G}$, select a model (φ, ψ) that minimizes the quantity $|\varphi(V(H)) \cap V(C^*)|$. Observe that if $\varphi(V(H)) \cap V(C^*) = \emptyset$, then φ is also a valid model in $\widehat{G} - \widehat{Y}$, contradicting that $\widehat{Y}$ is a solution to $\mathcal{F}$-Deletion on $\widehat{G}$. So in the remainder we consider the case that the minor model contains at least one vertex of C^*. We will build a minimal minor model of H in G using strictly fewer vertices of C^*, thereby contradicting the choice of (φ, ψ). Consider the X_0-labeled subgraph H'_L of $\widetilde{G}$ obtained by the following procedure, which is illustrated in Figure 2:

1. Start from the X_0-labeled subgraph of $\widetilde{G}$ induced by $\bigcup_{v \in V(H)} \varphi(v) \cap V(C^*)$, where each vertex u has labelset $N_G(u) \cap X_0$. As observed above, this subgraph is not empty.

2. Remove all edges from this subgraph, except those in the range of ψ and those that connect two vertices that belong to a common branch set under φ.
3. Contract every edge between two vertices that belong to a common branch set of φ, obtaining an X_0-labeled graph H'_L. (Recall that labelsets merge during edge contraction.)

Observe that H'_L has at most $|E(H)|$ edges, since each edge remaining in H'_L corresponds to an edge in the range of ψ. We claim that H'_L is an $n_{\mathcal{F}}$-restricted graph: the labelset of each vertex has size less than $n_{\mathcal{F}}$. To see this, observe that if some vertex of H'_L has a labelset $X' \subseteq X_0$ of size at least $n_{\mathcal{F}}$, then the pre-image of this vertex corresponds to a connected vertex subset A of $\varphi(V(H)) \cap V(C^*)$ such that $|N_G(A) \cap X_0| \geq n_{\mathcal{F}}$. Since (φ, ψ) is a minor model in $\widetilde{G} = G - (\hat{Y} \cup Y^*)$, this would imply that $C^*_L - Y^*$ has the one-vertex graph $I_{X'}$ with labelset X' as a labeled minor. But $I_{X'} \in \mathcal{Q}$ by (2), while Y^* breaks all labeled $\mathcal{Q}$-minors in C^*_L by definition; a contradiction. Hence H'_L is indeed $n_{\mathcal{F}}$-restricted.

Let H_L be an arbitrary connected component of H'_L. Since H_L is connected, $n_{\mathcal{F}}$-restricted, and contains at most $|E(H)|$ edges, we have $H_L \in \mathcal{H}_0$. As H_L clearly occurs as a labeled minor of $C^*_L - Y^*$, while Y^* breaks $\mathcal{Q}$ in C^*_L, we have $H_L \notin \mathcal{Q}$. By definition of $\mathcal{Q}$, this implies there are at least ρ connected components $C^1, \ldots, C^\rho$ of $\widehat{G} - X$ such that $C^i_L - \widehat{Y}$ contains H_L as X_0-labeled minor for each $i \in [\rho]$. By Lemma 8, the minimal model (φ, ψ) in $\widetilde{G}$ intersects at most $|X| + |V(H)| + |E(H)| \leq \rho$ components of $\widetilde{G} - X$ and therefore of $G - X$. Since $\varphi(V(H))$ also intersects $C^* \notin \{C^1, \ldots, C^\rho\}$, it follows that some C^i is disjoint from the range of (φ, ψ).

To finish the argument, fix C^i such that $\varphi(V(H)) \cap V(C^i) = \emptyset$ and $C^i_L - \widehat{Y}$ contains H_L as X_0-labeled minor. Let T denote the vertices of $\varphi(V(H)) \cap V(C^*)$ whose contraction in the process above resulted in the connected component H_L of H'_L. Then it is straightforward to verify that $G[(\varphi(V(H)) \setminus T) \cup (C^i - \widehat{Y})]$ contains H as a minor. The role that vertices of T played in the minor model (φ, ψ) can be replaced by the vertices of $C^i_L - \widehat{Y}$: each edge of ψ that was realized between vertices of T yielded an edge of H_L which is realized by a labeled H_L-minor in $C^i_L - \widehat{Y}$; each fragment of a branch set that was realized within C^* yielded a vertex of H_L that is realized in the H_L-minor in $C^i_L - \hat{Y}$; and finally the connectivity of the branch sets is ensured because the labeling ensures that for all fragments of branch sets in T that were adjacent to vertices of $X - \widehat{Y} = X_0$, the branch set of the H_L-minor in $C^i - \widehat{Y}$ realizing that fragment is also adjacent to all those vertices of X_0. Hence there is a *minimal* H-minor in $\widetilde{G}$ whose range is a subset of $(\varphi(V(H)) \setminus T) \cup (C^i - \widehat{Y})$. Since $T \subseteq C^*$ is not empty, this contradicts our choice of (φ, ψ) as a minimal H-model minimizing the intersection with C^*. ⌟

This concludes the proof of Lemma 10. ◀

5 Conclusion

Our goal in this paper was to obtain polynomial kernelizations for a wide range of graph problems, in terms of a rich class of structural parameterizations. We obtained polynomial kernelizations for $\mathcal{F}$-Deletion problems parameterized by a constant-treedepth modulator. The kernelization algorithm as presented here is only of theoretical interest. While the kernel size is polynomial for fixed $\mathcal{F}$ and η, the degree of the polynomial grows very quickly with $\mathcal{F}$ and η. It would be desirable to have a *uniformly polynomial* kernel size, of the form $f(\mathcal{F}, \eta)|X|^c$ for some constant c and function f. Unfortunately, Theorem 2 shows that even for the simplest choice of $\mathcal{F}$, corresponding to the Vertex Cover problem, the degree of the polynomial must depend exponentially on η and no uniformly polynomial kernelization exists. The bad news also extends in the other direction: when taking the simplest choice for η and working with a treedepth-one modulator (a vertex cover), the degree of the polynomial in the kernel size for $\mathcal{F}$-Deletion must depend on $\mathcal{F}$ [22, Theorem 1.1] and a uniformly-polynomial kernel does not exist.

References

1 Hans L. Bodlaender. Kernelization: New upper and lower bound techniques. In *Proc. 4th IWPEC*, pages 17–37, 2009. `doi:10.1007/978-3-642-11269-0_2`.

2 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. `doi:10.1016/j.jcss.2009.04.001`.

3 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. *J. ACM*, 63(5):44:1–44:69, 2016. `doi:10.1145/2973749`.

4 Marin Bougeret and Ignasi Sau. How much does a treedepth modulator help to obtain polynomial kernels beyond sparse graphs? *CoRR*, abs/1609.08095, 2016. URL: `http://arxiv.org/abs/1609.08095`.

5 Marin Bougeret and Ignasi Sau. How much does a treedepth modulator help to obtain polynomial kernels beyond sparse graphs? In *Proc. 12th IPEC (2017)*, volume 89, pages 10:1–10:13, 2018. `doi:10.4230/LIPIcs.IPEC.2017.10`.

6 Hubie Chen and Moritz Müller. One hierarchy spawns another: Graph deconstructions and the complexity classification of conjunctive queries. In *Proc. CSL-LICS 2014*, pages 32:1–32:10. ACM, 2014. `doi:10.1145/2603088.2603107`.

7 Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001. `doi:10.1006/jagm.2001.1186`.

8 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. `doi:10.1007/978-3-319-21275-3`.

9 Marek Cygan, Łukasz Kowalik, and Marcin Pilipczuk. Open problems from worker 2013, the workshop on kernels, April 2013. URL: `http://worker2013.mimuw.edu.pl/slides/worker-opl.pdf`.

10 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. On the hardness of losing width. *Theory Comput. Syst.*, 54(1):73–82, 2014. `doi:10.1007/s00224-013-9480-1`.

11 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014. `doi:10.1145/2629620`.

12 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. `doi:10.1007/978-1-4471-5559-1`.

13 Andrew Drucker. New limits to classical and quantum instance compression. In *Proc. 53rd FOCS*, pages 609–618, 2012. `doi:10.1109/FOCS.2012.71`.

14 Michael Elberfeld, Martin Grohe, and Till Tantau. Where first-order and monadic second-order logic coincide. *ACM Trans. Comput. Log.*, 17(4):25:1–25:18, 2016. `doi:10.1145/2946799`.

15 Henning Fernau. Kernelization, Turing kernels. In *Encyclopedia of Algorithms*, pages 1043–1045. Springer, 2016. `doi:10.1007/978-1-4939-2864-4_528`.

16 Fedor V. Fomin, Bart M. P. Jansen, and Michał Pilipczuk. Preprocessing subgraph and minor problems: When does a small vertex cover help? *J. Comput. Syst. Sci.*, 80(2):468–495, 2014. `doi:10.1016/j.jcss.2013.09.004`.

17 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Hitting forbidden minors: Approximation and kernelization. In *Proc. 28th STACS*, pages 189–200, 2011. doi:10.4230/LIPIcs.STACS.2011.189.

18 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar $\mathcal{F}$-Deletion: Approximation, kernelization and optimal FPT algorithms. In *Proc. 53rd FOCS*, pages 470–479, 2012. doi:10.1109/FOCS.2012.62.

19 Fedor V. Fomin and Torstein J. F. Strømme. Vertex cover structural parameterization revisited. In *Proc. 42nd WG*, volume 9941 of *LNCS*, pages 171–182, 2016. doi:10.1007/978-3-662-53536-3_15.

20 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011. doi:10.1016/j.jcss.2010.06.007.

21 Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. *J. Comput. Syst. Sci.*, 84:219–242, 2017. doi:10.1016/j.jcss.2016.09.002.

22 Archontia C. Giannopoulou, Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. Uniform kernelization complexity of hitting forbidden minors. *ACM Trans. Algorithms*, 13(3):35:1–35:35, 2017. doi:10.1145/3029051.

23 Jiong Guo and Rolf Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007. doi:10.1145/1233481.1233493.

24 Gregory Gutin. Kernelization, constraint satisfaction problems parameterized above average. In *Encyclopedia of Algorithms*, pages 1011–1013. Springer, 2016. doi:10.1007/978-1-4939-2864-4_524.

25 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited - Upper and lower bounds for a refined parameter. *Theory Comput. Syst.*, 53(2):263–299, 2013. doi:10.1007/s00224-012-9393-4.

26 Bart M. P. Jansen and Astrid Pieterse. Polynomial kernels for hitting forbidden minors under structural parameterizations. *CoRR*, abs/1804.08885, 2018. arXiv:1804.08885.

27 Bart M. P. Jansen, Venkatesh Raman, and Martin Vatshelle. Parameter ecology for feedback vertex set. *Tsinghua Science and Technology*, 19(4):387–409, 2014. doi:10.1109/TST.2014.6867520.

28 Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *ACM Trans. Algorithms*, 12(2):21:1–21:41, 2016. doi:10.1145/2797140.

29 Stefan Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113:58–97, 2014.

30 Stefan Kratsch. A randomized polynomial kernelization for vertex cover with a smaller parameter. In *Proc. 24th ESA*, volume 57 of *LIPIcs*, pages 59:1–59:17, 2016. doi:10.4230/LIPIcs.ESA.2016.59.

31 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *Proc. 53rd FOCS*, pages 450–459, 2012. doi:10.1109/FOCS.2012.46.

32 Diptapriyo Majumdar. Structural parameterizations of feedback vertex set. In *Proc. 11th IPEC*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:16, 2017. doi:10.4230/LIPIcs.IPEC.2016.21.

33 Diptapriyo Majumdar, Venkatesh Raman, and Saket Saurabh. Kernels for structural parameterizations of vertex cover - case of small degree modulators. In *Proc. 10th IPEC*, volume 43 of *LIPIcs*, pages 331–342, 2015. doi:10.4230/LIPIcs.IPEC.2015.331.

34 Neeldhara Misra. Kernelization, planar $\mathcal{F}$-Deletion. In *Encyclopedia of Algorithms*, pages 1033–1036. Springer, 2016. `doi:10.1007/978-1-4939-2864-4_527`.

35 G.L. Nemhauser and L.E. Trotter (jr.). Vertex packings: structural properties and algorithms. *Math. Program.*, 8:232–248, 1975. `doi:10.1007/BF01580444`.

36 J. Nešetřil and P. Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012. `doi:10.1007/978-3-642-27875-4`.

37 Michał Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. In *Proc. 33rd STACS*, volume 47 of *LIPIcs*, pages 57:1–57:15, 2016. `doi:10.4230/LIPIcs.STACS.2016.57`.

38 Felix Reidl, Peter Rossmanith, Fernando Sanchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In *Proc. 41st ICALP*, pages 931–942, 2014. `doi:10.1007/978-3-662-43948-7_77`.

39 Neil Robertson and Paul D. Seymour. Graph minors. V. Excluding a planar graph. *J. Comb. Theory, Ser. B*, 41(1):92–114, 1986. `doi:10.1016/0095-8956(86)90030-4`.

40 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2), 2010. `doi:10.1145/1721837.1721848`.

Quantum Algorithms for Connectivity and Related Problems

Michael Jarret
Perimeter Institute, Waterloo, ON, Canada
mjarret@perimeterinstitute.ca

Stacey Jeffery
Qusoft CWI, Amsterdam, The Netherlands
jeffery@cwi.nl

Shelby Kimmel
Middlebury College, Middlebury, VT, USA
skimmel@middlebury.edu

Alvaro Piedrafita
Qusoft CWI, Amsterdam, The Netherlands
piedrafita@cwi.nl

Abstract

An important family of span programs, *st*-connectivity span programs, have been used to design quantum algorithms in various contexts, including a number of graph problems and formula evaluation problems. The complexity of the resulting algorithms depends on the largest *positive witness size* of any 1-input, and the largest *negative witness size* of any 0-input. Belovs and Reichardt first showed that the positive witness size is exactly characterized by the effective resistance of the input graph, but only rough upper bounds were known previously on the negative witness size. We show that the negative witness size in an *st*-connectivity span program is exactly characterized by the *capacitance* of the input graph. This gives a tight analysis for algorithms based on *st*-connectivity span programs on any set of inputs.

We use this analysis to give a new quantum algorithm for estimating the capacitance of a graph. We also describe a new quantum algorithm for deciding if a graph is connected, which improves the previous best quantum algorithm for this problem if we're promised that either the graph has at least $\kappa > 1$ components, or the graph is connected and has small *average resistance*, which is upper bounded by the diameter. We also give an alternative algorithm for deciding if a graph is connected that can be better than our first algorithm when the maximum degree is small. Finally, using ideas from our second connectivity algorithm, we give an algorithm for estimating the *algebraic connectivity* of a graph, the second largest eigenvalue of the Laplacian.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Electrical networks, Quantum algorithms, Span programs, Connectivity, Graph theory

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.49

Acknowledgements SJ is supported by an NWO WISE Grant and NWO Veni Innovational Research Grant under project number 639.021.752. SK completed some of this work while at the Joint Center for Quantum Information and Computer Science (QuICS) at the University of Maryland. This research was supported in part by Perimeter Institute for Theoretical Physics. Research at Perimeter Institute is supported by the Government of Canada through Industry Canada and by the Province of Ontario through the Ministry of Economic Development and Innovation.

© Michael Jarret, Stacey Jeffery, Shelby Kimmel, and Alvaro Piedrafita;
licensed under Creative Commons License CC-BY

1 Introduction

Span programs are an algebraic model of computation first developed by Karchmer and Wigderson [10] to study classical logspace complexity, and introduced to the study of quantum algorithms by Reichardt and Špălek [15]. In [14, 12], Reichardt used the concept of span programs to prove that the general adversary bound gives a tight lower bound on the quantum query complexity of any given decision problem, thus showing the deep connection between span programs and quantum query algorithms.

Given a span program, a generic transformation compiles it into a quantum algorithm, whose query complexity is analyzed by taking the geometric mean of two quantities: the largest *positive witness size* of any 1-input; and the largest *negative witness size* of any 0-input. Thus, in order to analyze the query complexity of an algorithm obtained in this way, it is necessary to characterize, or at least upper bound, these quantities.

The relationship between quantum query algorithms and span programs is potentially a powerful tool, but this correspondence alone is not a recipe for finding such an algorithm, and producing an optimal span program for a given problem is generally difficult. Despite this difficulty, a number have been found for important problems such as k-distinctness [2], formula evaluation [15, 13], and st-connectivity [4]. The latter span program is of particular importance, as it has been applied to a number of graph problems [5], to generic formula evaluation problems [9], and underlies the learning graph framework [3]. The st-connectivity based algorithms are also of interest because, unlike with generic span program algorithms, it is often possible to analyze not only query complexity, but also the time complexity.

While span program algorithms are universal for quantum query algorithms, it can also be fruitful to analyze the unitaries used in these algorithms in ways that are different from how they appear in the standard span program algorithm. For example, Ref. [7] derives an algorithm to estimate span program witness sizes based on unitaries that appear in the span program algorithm. We will take a similar approach in this paper, deriving new algorithms based on unitaries that appear in the span program algorithm for st-connectivity.

The problems of st-connectivity and connectivity will be considered in this paper. For a family of undirected graphs G on N edges, for $N \in \mathbb{N}$, and vertex set containing s and t, the problem $st\text{-}\textsc{conn}_G$ is the following: Given $x \in \{0,1\}^{E(G)}$, decide if there is a path from s to t in $G(x)$, where $G(x)$ is the subgraph of G obtained by including an edge e if $x_e = 1$[1]. Similarly, the problem of $\textsc{conn}_G$ is the following: Given $x \in \{0,1\}^{E(G)}$, determine if every vertex in $G(x)$ is connected to every other vertex in $G(x)$.

1.1 Contributions

1. We provide a complete characterization of the query complexity of the st-connectivity span program algorithm. We do this by showing that the negative witness size of the st-connectivity span program is exactly the *effective capacitance* of the input graph. (The positive witness size for this span program was previously known to be exactly the *effective resistance* [4, 9].) The *effective capacitance* is a measure that depends on the size and number of cuts between s and t (in the case they are disconnected), and is commonly used to analyze electrical networks of capacitors. This characterization tells us that quantum algorithms can quickly decide st-connectivity on graphs that are promised to have either small effective resistance or small effective capacitance.

[1] We can consider more complicated ways of associating edges with input variables in 2.2, but the basic idea is captured by this simpler picture.

2. We describe a new quantum algorithm for estimating the effective capacitance of an input graph $G(x)$ to multiplicative error ε, with complexity $\widetilde{O}(\varepsilon^{-3/2}\sqrt{C_{s,t}(G(x))p})$, where $C_{s,t}(G(x))$ is the effective-capacitance between s and t in $G(x)$, and p is the length of the longest self-avoiding st-path in G.
3. We create and analyze a new algorithm for $\textsc{conn}_G$. Previously, for a graph with n vertices, an optimal $\widetilde{O}(n^{3/2})$ upper bound on the time complexity of this problem was known [6], and an optimal span-program-based quantum algorithm was presented by Āriņš [1], which also uses only $O(\log n)$ space. If R upper bounds the average resistance of any connected input, all disconnected inputs have at least $\kappa > 1$ components, and U is the cost of taking a step of a quantum walk on G then our algorithm has the following properties:
   - For graphs without multi-edges, our algorithm has query complexity $O(n\sqrt{R/\kappa})$ and time complexity $\widetilde{O}(n\sqrt{R/\kappa}\mathsf{U})$.
   - For graphs with multi-edges, our algorithm has query complexity $O(\frac{n^{3/4}\sqrt{Rd_{\max}(G)}}{\kappa^{1/4}})$, where $d_{\max}(G)$ is the maximum degree of any vertex in the graph, and time complexity $\widetilde{O}(n^{3/4}\sqrt{Rd_{\max}(G)}/\kappa^{1/4}\mathsf{U})$.
   - Our algorithm uses $O(\log n)$ space.
   - Our algorithm is the first connectivity algorithm to explicity apply to cases where G is not necessarily the complete graph.
   - In the worst case, our algorithm achieves the optimal query complexity of $O(n^{3/2})$.
4. We present an alternative approach to deciding graph connectivity using phase estimation on a unitary derived from the st-connectivity span program. This phase estimation uses a different initial state from that used in the span program algorithm. We first show that the quantum query complexity of deciding $\textsc{conn}_G$ is $O(\sqrt{nd_{\max}(G)/(\kappa\lambda)})$, when either $G(x)$ is connected and the second smallest eigenvalue of the Laplacian of $G(x)$, $\lambda_2(G(x))$, is at least λ, or $G(x)$ has at least $\kappa > 1$ connected components. We are able to give time-efficient versions of our second algorithm in two contexts:
   a. Under the promise that if $G(x)$ is connected, then $\lambda_2(G(x)) \geq \lambda$, and otherwise $G(x)$ has at least κ connected components, we can solve $\textsc{conn}_G$ in time complexity $\widetilde{O}\left(\sqrt{\frac{nd_{\text{avg}}(G)}{\kappa\lambda_2(G)}}\left(\mathsf{S}+\sqrt{\frac{d_{\max}(G)}{\lambda}}\mathsf{U}\right)\right)$, where U is the complexity of implementing a step of a quantum walk on G, S is the cost generating a quantum state corresponding to the stationary distribution of a random walk on G, and $d_{\text{avg}}(G)$ is the average degree of the vertices of G.
   b. When G is a Cayley graph of degree d, the time complexity is upper bounded by $\widetilde{O}\left(\sqrt{\frac{nd}{\kappa\lambda}}\mathsf{U}+\sqrt{\frac{nd}{\kappa\lambda_2(G)}}\Lambda\right)$, where Λ is the cost of computing the eigenvalues of G. This gives an upper bound of $\widetilde{O}(n/\sqrt{\lambda\kappa})$ when G is a complete graph, and $\widetilde{O}(\sqrt{n/(\lambda\kappa)})$ when G is a Boolean hypercube.
5. We give an algorithm to estimate the *algebraic connectivity* of $G(x)$, $\lambda_2(G(x))$, when G is a complete graph. The algebraic connectivity is closely related to the inverse of the mixing time, which is known to be small for many interesting families of graphs such as expander graphs. We give a protocol that with probability at least $2/3$ outputs an estimate of $\lambda_2(G(x))$ up to multiplicative error ε in time complexity $\widetilde{O}\left(\frac{1}{\varepsilon}\frac{n}{\sqrt{\lambda_2(G(x))}}\right)$.

1.2 Open Problems

Our work suggests several directions for new research. Since st-connectivity is fairly ubiquitous, it seems that our approach may, in turn, help analyze applications of *st*-connectivity. Additionally, we provide two algorithms for deciding connectivity, items 3 and 4 above. At least naively, it seems like our two algorithms are incomparable, even though they are based on similar unitaries. It would be worthwhile to understand whether the two approaches are fundamentally different. Finally, it would be interesting to see whether one can extend our algorithm for estimating algebraic connectivity to accept more general parent graphs than the complete graph.

2 Preliminaries

2.1 Linear Algebra Notation

For a subspace V of some inner product space, let Π_V denote the orthogonal projector onto V. For a linear operator A, let $\sigma_{\min}(A)$ (respectively $\sigma_{\max}(A)$) denote its smallest (resp. largest) non-zero singular value. Let $\ker A$ denote the kernel of A, $\mathrm{row}(A)$ denote the rowspace of A, and $\mathrm{col}(A)$ the columnspace of A. For a unitary U with eigenvalues $e^{i\theta_1},\dots,e^{i\theta_N}$, let $\Delta(U)=\min\{|\theta_i|:\theta_i\neq 0\}$ denote the *phase gap* of U.

2.2 Graph Theory

We will consider multigraphs, so we refer to each edge in the graph using its endpoints and a unique label ℓ, as, for example: $(\{u,v\},\ell)$. The label ℓ uniquely specifies the edge, but we include the endpoints for convenience. Let $\overrightarrow{E}(G)=\{(u,v,\ell):(\{u,v\},\ell)\in E(G)\}$ be the directed edges of G. Furthermore, for any set of edges E, we let $\overrightarrow{E}=\{(u,v,\ell):(\{u,v\},\ell)\in E\}$ represent the corresponding set of directed edges. We will sometimes write (u,v,ℓ) for an undirected edge, but when talking about undirected edges, we have $(u,v,\ell)=(v,u,\ell)$.

For $x\in\{0,1\}^{E(G)}$, we define $G(x)$ as the subgraph of G in which $e\in E(G)$ is included if and only if $x_e=1$. In general there can be a more complicated association between the edges of G and literals x_i and $\bar{x}_i$, but for simplicity, we don't make this explicit.

A *network* $\mathcal{N}=(G,c)$ consists of a graph G combined with a positive real-valued *weight* function $c:E(G)\longrightarrow\mathbb{R}^+$. Since c is a map on undirected edges, we can easily extend it to map on directed edges such that $c(u,v,\ell)=c(v,u,\ell)$, and we overload our notation accordingly. We will often assume that some c is implicit for a graph G and let $\mathcal{A}_G=\sum_{(u,v,\ell)\in E(G)}c(u,v,\ell)(|u\rangle\langle v|+|v\rangle\langle u|)$ denote its weighted adjacency matrix. Note that $\mathcal{A}_G$ only depends on the total weight of edges from u to v, and is independent of the number of edges across which this weight is distributed. Let $d_G(u)=\sum_{v,\ell:(u,v,\ell)\in E(G)}c(u,v,\ell)$ denote the weighted degree of u in G, under the implicit weight function c, and let $d_{\max}(G)=\max_{u\in V(G)}d_G(u)$. Let $\mathcal{D}_G=\sum_{u\in V(G)}d_G(u)|u\rangle\langle u|$ denote the weighted degree matrix, and let $L_G=\mathcal{D}_G-\mathcal{A}_G$ denote the Laplacian of G. The Laplacian is always positive semidefinite, so its eigenvalues are real and non-negative. For $|\mu\rangle=\sum_{u\in V(G)}|u\rangle$, it is always the case that $L_G|\mu\rangle=0$, so the smallest eigenvalue of L_G is 0. Let $\lambda_2(G)$ denote the second smallest eigenvalue of L_G, including multiplicity. This value is called the *algebraic connectivity* or the *Fiedler value* of G, and it is non-zero if and only if G is connected.

Consider a graph G with specially labeled vertices s and t that are connected in G. An *st-flow* is any linear combination of *st*-paths. More precisely:

▶ **Definition 1** (Unit st-flow and energy)**.** Let G be an undirected graph with $s, t \in V(G)$, and s and t connected. Then a *unit st-flow* on G is a function $\theta : \overrightarrow{E}(G) \to \mathbb{R}$ such that:

1. For all $(u, v, \ell) \in \overrightarrow{E}(G)$, $\theta(u, v, \ell) = -\theta(v, u, \ell)$;
2. $\sum_{v,\ell:(s,v,\ell)\in\overrightarrow{E}(G)} \theta(s, v, \ell) = \sum_{v,\ell:(v,t,\ell)\in\overrightarrow{E}(G)} \theta(v, t, \ell) = 1$; and
3. for all $u \in V(G) \setminus \{s, t\}$, $\sum_{v,\ell:(u,v,\ell)\in\overrightarrow{E}(G)} \theta(u, v, \ell) = 0$.

Given an implicit weighting c, the *unit flow energy* of θ on $E' \subseteq E(G(x))$, is $J_{E'}(\theta) = \frac{1}{2}\sum_{e\in\overrightarrow{E'}} \frac{\theta(e)^2}{c(e)}$.

▶ **Definition 2** (Effective resistance and average resistance)**.** Let G be a graph with implicit weighting c and $s, t \in V(G)$. If s and t are connected in $G(x)$, the *effective resistance* of $G(x)$ between s and t is $R_{s,t}(G(x)) = \min_\theta J_{E(G(x))}(\theta)$, where θ runs over all unit st-unit flows of $G(x)$. If s and t are not connected in $G(x)$, $R_{s,t}(G(x)) = \infty$. For a connected graph G, we can define the *average resistance* by $R_{\text{avg}}(G) := \frac{1}{n(n-1)}\sum_{s,t\in V:s\neq t} R_{s,t}(G)$.

Intuitively, $R_{s,t}$ characterizes "how connected" the vertices s and t are in a network. The more, shorter paths connecting s and t, and the more weight on those paths, the smaller the effective resistance. We next introduce a measure of how disconnected s and t are, in the case that we are considering a subgraph $G(x)$ of G where s and t are not connected.

▶ **Definition 3** (Unit st-potential)**.** Let G be an undirected weighted graph with $s, t \in V(G)$, and s and t connected. For $G(x)$ such that s and t are not connected, a *unit st-potential* on $G(x)$ is a function $\mathcal{V} : V(G) \to \mathbb{R}^+$ such that $\mathcal{V}(s) = 1$ and $\mathcal{V}(t) = 0$ and $\mathcal{V}(u) = \mathcal{V}(v)$ if $(u, v, \ell) \in E(G(x))$.

A unit st-potential is a witness of the disconnectedness of s and t in $G(x)$, which generalizes the notion of an st-cut. (An st-cut is a unit potential that only takes values 0 and 1.)

▶ **Definition 4** (Unit Potential Energy)**.** Given a graph G with implicit weighting c and a unit st-potential $\mathcal{V}$ on $G(x)$, the *unit potential energy* of $\mathcal{V}$ on $E' \subseteq E(G)$ is defined $\mathcal{J}_{E'}(\mathcal{V}) = \frac{1}{2}\sum_{(u,v,\ell)\in\overrightarrow{E'}}(\mathcal{V}(u) - \mathcal{V}(v))^2 c(u, v, \ell)$.

▶ **Definition 5** (Effective capacitance)**.** Let G be a graph with implicit weighting c and $s, t \in V(G)$. If s and t are not connected in $G(x)$, the *effective capacitance* between s and t of $G(x)$ is $C_{s,t}(G(x)) = \min_{\mathcal{V}} \mathcal{J}_{E(G)}(\mathcal{V})$, where $\mathcal{V}$ runs over all unit st-potentials on $G(x)$. If s and t are connected, $C_{s,t}(G(x)) = \infty$.

In physics, capacitance measures how well a system of two separated conductors stores electric charge. The ratio of the amount of stored charge to the voltage difference between the conductors is a constant that depends only on the geometry of the set-up. This ratio is called the effective capacitance. We discuss this intuition further in Section 2.2 and Appendix A of [8].

Connectivity and st-connectivity

We will consider problems parametrized by a parent graph G, by which we more precisely mean a family of graphs $\{G_n\}_{n\in\mathbb{N}}$ where G_n is a graph on n vertices. We will generally drop the subscript n.

A graph is connected if there is a path between every pair of vertices. For a family of graphs G, and $X \subseteq \{0, 1\}^{E(G)}$, $\textsc{conn}_{G,X}$ is the *connectivity problem*, defined for all $x \in X$ by $\textsc{conn}_{G,X}(x) = 1$ if $G(x)$ is connected, and $\textsc{conn}_{G,X}(x) = 0$ if $G(x)$ is not connected.

Similarly, for $s, t \in V(G)$, defined st-$\textsc{conn}_{G,X}$ by st-$\textsc{conn}_{G,X}(x) = 1$ if there is a path from s to t in $G(x)$, and st-$\textsc{conn}_{G,X}(x) = 0$ otherwise, for all $x \in X$.

We will consider CONN and st-CONN in the edge-query input model, meaning that we have access to a standard quantum oracle O_x, defined $O_x|i\rangle|b\rangle = |i\rangle|b \oplus x_i\rangle$, where x_i is the i^{th} bit of i. Since every edge of G is associated with an input variable, as described in 2.2, for any edge in G, we can check if it is also present in $G(x)$ using one query to O_x.

2.3 Span Programs and Witness Sizes

Span programs [10] were introduced to quantum algorithms by Reichardt and Špalek [15], and have since proven to be important for designing quantum algorithms in the query model.

▶ **Definition 6** (Span Program). A span program $P = (H, U, \tau, A)$ on $\{0,1\}^N$ is made up of **(I)** finite-dimensional inner product spaces $H = H_1 \oplus \cdots \oplus H_N$, and $\{H_{j,b} \subseteq H_j\}_{j\in[N],b\in\{0,1\}}$ such that $H_{j,0} + H_{j,1} = H_j$, **(II)** a vector space U, **(III)** a non-zero *target vector* $\tau \in U$, and **(IV)** a linear operator $A : H \to U$. For every string $x \in \{0,1\}^N$, we associate the subspace $H(x) := H_{1,x_1} \oplus \cdots \oplus H_{N,x_N}$, and an operator $A(x) := A\Pi_{H(x)}$.

▶ **Definition 7** (Positive and Negative Witness). Let P be a span program on $\{0,1\}^N$ and let x be a string $x \in \{0,1\}^N$. Then we call $|w\rangle$ a *positive witness for x in P* if $|w\rangle \in H(x)$, and $A|w\rangle = \tau$. We define the *positive witness size of x* as:

$$w_+(x, P) = w_+(x) = \min\{\||w\rangle\|^2 : |w\rangle \in H(x), A|w\rangle = \tau\}, \tag{1}$$

if there exists a positive witness for x, and $w_+(x) = \infty$ otherwise.

Let $\mathcal{L}(U, \mathbb{R})$ denote the set of linear maps from U to $\mathbb{R}$. We call a linear map $\omega \in \mathcal{L}(U, \mathbb{R})$ a *negative witness for x in P* if $\omega A\Pi_{H(x)} = 0$ and $\omega\tau = 1$. We define the *negative witness size of x* as:

$$w_-(x, P) = w_-(x) = \min\{\|\omega A\|^2 : \omega \in \mathcal{L}(U, \mathbb{R}), \omega A\Pi_{H(x)} = 0, \omega\tau = 1\}, \tag{2}$$

if there exists a negative witness, and $w_-(x) = \infty$ otherwise. If $w_+(x)$ is finite, we say that x is *positive* (wrt. P), and if $w_-(x)$ is finite, we say that x is *negative.* We let P_1 denote the set of positive inputs, and P_0 the set of negative inputs for P.

For a function $f : X \to \{0,1\}$, with $X \subseteq \{0,1\}^N$, we say P *decides* f if $f^{-1}(0) \subseteq P_0$ and $f^{-1}(1) \subseteq P_1$. Given a span program P that decides f, one can use it to design a quantum algorithm whose output is $f(x)$ (with high probability), given access to the input $x \in X$ via queries of the form $\mathcal{O}_x : |i, b\rangle \mapsto |i, b \oplus x_i\rangle$.

The following theorem is due to [12] (see [7] for a version with similar notation).

▶ **Theorem 8.** *Let $U(P,x) = (2\Pi_{\ker A} - I)(2\Pi_{H(x)} - I)$. Fix $X \subseteq \{0,1\}^N$ and $f : X \to \{0,1\}$, and let P be a span program on $\{0,1\}^N$ that decides f. Let $W_+(f,P) = \max_{x\in f^{-1}(1)} w_+(x,P)$ and $W_-(f,P) = \max_{x\in f^{-1}(0)} w_-(x,P)$. Then there is a bounded error quantum algorithm that decides f by making $O(\sqrt{W_+(f,P)W_-(f,P)})$ calls to $U(P,x)$, and elementary gates. In particular, this algorithm has quantum query complexity $O(\sqrt{W_+(f,P)W_-(f,P)})$.*

Ref. [7] defines the *approximate positive witness size*, $\tilde{w}_+(x,P)$ as the smallest $\||w\rangle\|^2$ such that $A|w\rangle = \tau$ and $\|\Pi_{H(x)^\perp}|w\rangle\|$, rather than being required to be 0, should be as small as possible. In particular, every x has a finite approximate positive witness size, not only those in P_1.

▶ **Theorem 9** ([7]). *Let $U(P,x) = (2\Pi_{\ker A} - I)(2\Pi_{H(x)} - I)$. Fix $X \subseteq \{0,1\}^N$ and $f : X \to \mathbb{R}_{\geq 0}$. Let P be a span program on $\{0,1\}^N$ such that for all $x \in X$, $f(x) = w_-(x,P)$ and define $\widetilde{W}_+ = \widetilde{W}_+(P) = \max_{x\in X} \tilde{w}_+(x,P)$. Then there is a quantum algorithm that estimates f to accuracy ϵ and that uses $\widetilde{O}\left(\epsilon^{-3/2}\sqrt{w_-(x)\widetilde{W}_+}\right)$ calls to $U(P,x)$ and elementary gates.*

A span program for st-connectivity

An important example of a span program is one for st-connectivity, first introduced in [10], and used in [4] to give a new quantum algorithm for st-connectivity. Given some implicit weighting function c on G, the span program is as follows, which we denote P_G:

$$\begin{aligned} &\forall e \in E(G), H_{e,1} = \mathrm{span}\{|u,v,\ell\rangle, |v,u,\ell\rangle : e = (\{u,v\},\ell)\} && U = \mathrm{span}\{|v\rangle : v \in V(G)\} \\ &\forall e = (u,v,\ell) \in \overrightarrow{E}(G) :\; A|u,v,\ell\rangle = \sqrt{c(u,v,\ell)}(|u\rangle - |v\rangle) && \tau = |s\rangle - |t\rangle \end{aligned} \tag{3}$$

If s and t are connected in $G(x)$, then a linear combination of weighted st-paths in $G(x)$ is a positive witness for x. Furthermore, this is the only possible positive witness form, so x is a positive input for P_G if and only if $G(x)$ is st-connected, and in particular, $w_+(x, P_G) = \frac{1}{2}R_{s,t}(G(x))$ [4]. Since the weights $c(e)$ are positive, the set of positive inputs of P_G are independent of the choice of c, however, the witness sizes will depend on c.

3 Effective Capacitance and st-connectivity

In [8, Theorem 17], we prove the following theorem, exactly characterizing the negative witness size of the st-connectivity span program:

▶ **Theorem 10.** *Let P_G be the span program in Eq.* (3)*. Then for any $x \in \{0,1\}^N$, $w_-(x, P_G) = 2C_{s,t}(G(x))$.*

Previously, the negative witness size of P_G was characterized by the size of a cut [15] or, in planar graphs, the effective resistance of a graph related to the planar dual of $G(x)$ [9].

As a corollary of Theorem 10, we have the following:

▶ **Theorem 11.** *Let G be a multigraph with $s,t \in V(G)$. Then for* any *choice of (non-negative, real-valued) implicit weight function, the bounded error quantum query complexity of evaluating $st\text{-}\mathrm{CONN}_{G,X}$ is*

$$O\left(\sqrt{\max_{x \in X : st\text{-}\mathrm{CONN}_{G,X}(x)=1} R_{s,t}(G(x)) \times \max_{x \in X : st\text{-}\mathrm{CONN}_{G,D}(x)=0} C_{s,t}(G(x))}\right). \tag{4}$$

Proof. This follows from Theorem 10 and the fact that $w_+(x, P_G) = \frac{1}{2}R_{s,t}(G(x))$, proven in [4] and generalized to the weighted case in [9]. Then Theorem 8 gives the result. ◀

We emphasize that Theorem 11 holds for $R_{s,t}$ and $C_{s,t}$ defined with respect to any weight function, some of which may give a significantly better complexity for solving this problem.

3.1 Estimating the Capacitance of a Circuit

By Theorem 10, $w_-(x, P_G) = 2C_{s,t}(G(x))$, so we can apply Theorem 9 to estimate $C_{s,t}(G(x))$. By Theorem 9, the complexity of doing this depends on $C_{s,t}(G(x))$ and $\widetilde{W}_+(P_G) = \max_x \tilde{w}_+(x, P_G)$. We prove the following theorem in [8]:

▶ **Theorem 12.** *For the span program P_G, we have that $\widetilde{W}_+(P_G) = O(\max_p J_{E(G)}(p))$, where the maximum runs over all st-unit flows p that are paths from s to t.*

To prove 12, we first relate unit st-flows on G to approximate positive witnesses. Intuitively, an approximate positive witness is an st-flow on G that has energy as small as possible on edges in $E(G) \setminus E(G(x))$. Thus, we can upper bound the approximate positive witness size by the highest possible energy of any st-flow on G, which is always achieved by a flow that is an st-path. Note that when the weights are all 1, $\max_p J_E(G)(p)$ is just the length of the longest self-avoiding st-path in G. Combining Theorems 10, 12 and 9, we have:

▶ **Corollary 13.** *Given a network (G, c), with $s, t \in V(G)$ and access to an oracle O_x, the bounded error quantum query complexity of estimating $C_{s,t}(G(x))$ to accuracy ϵ is $\widetilde{O}(\epsilon^{-3/2}\sqrt{C_{s,t}(G(x))\max_p J_{E(G)}(p)})$ where the maximum runs over all st-unit flows p that are paths from s to t.*

▶ **Corollary 14.** *Let U be the cost of implementing $|u\rangle|0\rangle \mapsto \sum_{(u,v,\ell)\in\overrightarrow{E}(G)} \sqrt{\frac{c(u,v,\ell)}{d_G(u)}}|u, v, \ell\rangle$. Then the quantum time complexity of estimating $C_{s,t}(G(x))$ to accuracy ϵ is*

$$\widetilde{O}\left(\epsilon^{-3/2}\sqrt{C_{s,t}(G(x))\max_p J_{E(G)}(p)\mathsf{U}}\right).$$

Proof. By [9] (generalizing [4]), $U(P_G, x)$, from Theorem 9, can be implemented in cost $O(\mathsf{U})$. ◀

3.2 Deciding Connectivity

Note that $\text{CONN}_{G,X} = \bigwedge_{\{u,v\}:u,v\in V(G)} uv\text{-CONN}_{G,X}$. Thus connectivity is equivalent [11, 9] to $n(n-1)/2$ st-connectivity problems in series, one for each pair of distinct vertices in $V(G)$. (Ref. [1] uses a similar approach, but only looks at $n-1$ instances — the pairs s and v for each $v \in V(G)$. Our approach is symmetrized over the vertices, so the analysis is simpler.)

More precisely, we define a graph $\mathcal{G}$ such that:

$$V(\mathcal{G}) = V(G) \times \{\{u, v\} : u \neq v \in V(G)\}, \quad E(\mathcal{G}) = E(G) \times \{\{u, v\} : u \neq v \in V(G)\} \quad (5)$$

where $\times$ denotes the Cartesian product. We think of the $\{u, v\}$ terms in (5) as an extra label denoting that that edge or vertex is in the $\{u, v\}^{\text{th}}$ copy of the graph G present as a subgraph in $\mathcal{G}$. Choose any labeling of the vertices from 1 to n (with slight abuse of notation, we use u both for the original vertex name and the label). We then label the vertex $(1, \{1, 2\})$ as s and the vertex $(n, \{n-1, n\})$ as t. Next identify vertices $(v, \{u, v\})$ and $(u, \{u, v+1\})$ if $u < v$ and $v < n$, and identify vertices $(v, \{u, v\})$ and $(u+1, \{u+1, u+2\})$ if $v = n$ and $u < n-1$. See [8] for a graphical example of this construction.

Finally, we define $\mathcal{G}(x)$ to be the subgraph of $\mathcal{G}$ with edges $E(\mathcal{G}(x)) = E(G(x)) \times \{\{u, v\} : u \neq v \in V(G)\}$. Clearly, any st-path in $\mathcal{G}(x)$ must go through each of the copies of $G(x)$, meaning it must include, for each $\{u, v\}$, a uv-path through the copy of $G(x)$ labeled $\{u, v\}$. Thus, there is an st-path in $\mathcal{G}(x)$ if and only if $G(x)$ is connected.

We consider the span program $P_{\mathcal{G}}$, where $c(e) = 1$ for all $e \in E(\mathcal{G})$. We will use $P_{\mathcal{G}}$ to solve st-connectivity on $\mathcal{G}(x)$. To analyze the resulting algorithm, we need to upper bound the negative and positive witness sizes $w_-(x, P_{\mathcal{G}}) = 2C_{s,t}(\mathcal{G}(x))$ and $w_+(x, P_{\mathcal{G}}) = \frac{1}{2}R_{s,t}(\mathcal{G}(x))$. Using the rule that resistances in series add, we get:

▶ **Lemma 15.** *For any x such that $G(x)$ is connected, $w_+(x, P_{\mathcal{G}}) = \frac{n(n-1)}{2}R_{\text{avg}}(G(x))$.*

In [8, Lemma 24], we bound $C_{s,t}(\mathcal{G}(x))$, to prove the following:

▶ **Lemma 16.** *Fix $\kappa > 1$, and suppose $G(x)$ has κ connected components. Then if G is a subgraph of a complete graph (that is, G has at most one edge between any pair of vertices), we have $w_-(x, P_{\mathcal{G}}) = O(1/\kappa)$. Otherwise, we have $w_-(x, P_{\mathcal{G}}) = O(d_{\max}(G)/\sqrt{n\kappa})$.*

Combining Lemmas 16 and 15 with Theorem 8, we have the following:

▶ **Theorem 17.** *For any family of graphs G such that G is a subgraph of a complete graph, and $X \subseteq \{0,1\}^{E(G)}$ such that for all $x \in X$, if $G(x)$ is connected, $R_{\mathrm{avg}}(G(x)) \leq R$, and if $G(x)$ is not connected, it has at least κ components, the bounded error quantum query complexity of* $\textsc{conn}_{G,X}$ *is* $O\left(n\sqrt{R/\kappa}\right)$.

For any family of connected graphs G and $X \subseteq \{0,1\}^{E(G)}$ such that for all $x \in X$, if $G(x)$ is connected, $R_{\mathrm{avg}}(G(x)) \leq R$, and if $G(x)$ is not connected, it has at least κ components, the bounded error quantum query complexity of $\textsc{conn}_{G,X}$ *is* $O\left(n^{3/4}\sqrt{Rd_{\max}(G)}/\kappa^{1/4}\right)$.

▶ **Corollary 18.** *Let* U *be the cost of implementing* $|u\rangle|0\rangle \mapsto \sum_{(u,v,\ell)\in\overrightarrow{E}(G)} \sqrt{d_G^{-1}(u)}|u,v,\ell\rangle$. *If G is subset of a complete graph, the quantum time complexity of* $\textsc{conn}_{G,X}$ *is* $O(n\sqrt{R/\kappa}\mathsf{U})$. *For any family of connected graphs G and $X \subseteq \{0,1\}^{E(G)}$ such that for all $x \in X$, if $G(x)$ is connected, $R_{\mathrm{avg}}(G(x)) \leq R$, and if $G(x)$ is not connected, it has at least κ components, the quantum time complexity of* $\textsc{conn}_{G,X}$ *is* $O\left(n^{3/4}\sqrt{Rd_{\max}(G)}/\kappa^{1/4}\mathsf{U}\right)$.

Proof. By [9] (generalizing [4]), $U(P_{\mathcal{G}}, x)$ can be implemented in cost $O(\mathsf{U})$. ◀

4 Spectral Algorithm for Deciding Connectivity

In this section, we give alternative quantum algorithms for connectivity. We first present an algorithmic template, outlined in Algorithm 25, that requires the instantiation of a certain initial state. Since this initial state is independent of the input, we already get an upper bound on the quantum query complexity, as follows:

▶ **Corollary 19.** *Fix any $\lambda > 0$ and $\kappa > 1$. For any family of connected graphs G and $X \subseteq \{0,1\}^{E(G)}$ such that for all $x \in X$, either $\lambda_2(G(x)) \geq \lambda$ or $G(x)$ has at least κ connected components, the bounded error quantum query complexity of* $\textsc{conn}_{G,X}$ *is* $O\left(\sqrt{\frac{nd_{\max}(G)}{\kappa\lambda}}\right)$.

In [8, Section 5.1], we describe one such initial state, and how to prepare it, leading to the following upper bound, in which U is the cost of performing one step of a quantum walk on G, and S is the cost of preparing a quantum state corresponding to the stationary distribution of a quantum walk on G:

▶ **Theorem 20.** *Fix any $\kappa > 1$ and $\lambda > 0$. For any family of connected graphs G and $X \subseteq \{0,1\}^{E(G)}$ such that $\forall x \in X$, either $\lambda_2(G(x)) \geq \lambda$, or $G(x)$ has at least κ components,* $\textsc{conn}_{G,X}$ *can be solved in bounded error in time* $\widetilde{O}\left(\sqrt{\frac{nd_{\mathrm{avg}}(G)}{\kappa\lambda_2(G)}}\left(\mathsf{S} + \sqrt{\frac{d_{\max}(G)}{\lambda}}\mathsf{U}\right)\right)$.

In [8, Section 5.2], we restrict our attention to the case where G is a Cayley graph, and give an alternative instantiation of Algorithm 25, proving the following, where Λ is the cost of computing the eigenvalues of G:

▶ **Theorem 21.** *Fix any $\lambda > 0$ and integer $\kappa > 1$. For any family of connected graphs G such that each G is a Cayley graph over an Abelian group and $X \subseteq \{0,1\}^{E(G)}$ such that for all $x \in X$, either $\lambda_2(G(x)) \geq \lambda$ or $G(x)$ has at least κ components,* $\textsc{conn}_{G,X}$ *can be solved in bounded error in time* $\widetilde{O}\left(\sqrt{\frac{nd}{\kappa\lambda}}\mathsf{U} + \sqrt{\frac{nd}{\kappa\lambda_2(G)}}\Lambda\right)$.

The results in this section, in contrast to the previous connectivity algorithm, apply with respect to any weighting of the edges of G. Applying non-zero weights to the edges of G does not change which subgraphs $G(x)$ are connected, but it does impact the complexity of our algorithm. Thus, we get algorithms with the complexities given in Corollary 19 and Theorems 20 and 21, where $d_{\max}(G)$ and $d_{\text{avg}}(G)$ are in terms of weighted degrees, and $\lambda_2(G)$ and $\lambda_2(G(x))$ are in terms of weighted Laplacians.

Finally, in 4.1, we describe how when G is a complete graph, these ideas can be used to design algorithms, not only for deciding connectivity, but also for estimating the *algebraic connectivity* of a graph, a measure of how connected a graph is. In particular, we show:

▶ **Theorem 22.** *Let G be the complete graph on n vertices. There exists a quantum algorithm that, on input x, with probability at least 2/3, outputs an estimate $\tilde{\lambda}$ such that $\left|\tilde{\lambda} - \lambda_2(G(x))\right| \leq \varepsilon\lambda_2(G(x))$, where $\lambda_2(G(x))$ is the algebraic connectivity of $G(x)$, in complexity $\widetilde{O}\left(n/\varepsilon\sqrt{\lambda_2(G(x))}\right)$.*

Let $P_G = (H, U, A, \tau)$ be the span program for *st*-connectivity defined in 3. Note that only τ depends on s and t, and we will not be interested in τ here. We let $A(x) = A\Pi_{H(x)}$. A simple calculation gives $A(x)A(x)^T = 2L_{G(x)}$ and $AA^T = 2L_G$, where $L_{G(x)}$ and L_G are the Laplacians of $G(x)$ and G respectively. Recall that for any G, the eigenvalues of L_G lie in $[0, d_{\max}]$, with $|\mu\rangle = \frac{1}{\sqrt{n}}\sum_v |v\rangle$ as a 0-eigenvalue. In our case, since G is assumed to be connected, $|\mu\rangle$ is the only 0-eigenvector of L_G, so $\text{row}(L_G)$ is the orthogonal complement of $|\mu\rangle$. For any x, $G(x)$ also has $|\mu\rangle$ as a 0-eigenvalue, but if $G(x)$ is connected, this is the only 0-eigenvalue. In general, the dimension of the 0-eigenspace of $L_{G(x)}$ is the number of components of $G(x)$. Thus, we have the following.

- The multiset of nonzero eigenvalues of L_G are exactly half of the squared singular values of A, and in particular, since no eigenvalue of L_G can be larger than the maximum degree of G, $\sigma_{\max}(A) \leq \sqrt{2d_{\max}(G)}$.
- The multiset of nonzero eigenvalues of $L_{G(x)}$ are exactly half the squared singular values of $A(x)$, and if $G(x)$ is connected, then $\sigma_{\min}(A(x)) = \sqrt{2\lambda_2(G(x))}$, where $\lambda_2(G(x))$ is the second smallest eigenvalue of $L_{G(x)}$, which is non-zero if and only if $G(x)$ is connected.
- The support of L_G is $\text{col}(A)$, which is the orthogonal subspace of $|\mu\rangle = \frac{1}{\sqrt{n}}\sum_v |v\rangle$.

For a particular span program P, and input x, an associated unitary $U(P, x) = (2\Pi_{\ker A} - I)(2\Pi_{H(x)} - I)$ can be used to construct quantum algorithms, for example, for deciding the span program. Then by [7, Theorem 3.10], which states that $\Delta(U(P, x)) \geq 2\sigma_{\min}(A(x))/\sigma_{\max}(A)$, we have the following.

▶ **Lemma 23.** *Let P_G be the st-connectivity span program from 3. Then $\Delta(U(P, x)) \geq 2\sqrt{\lambda_2(G(x))/d_{\max}(G)}$.*

Our algorithm will be based on the following connection between the connectivity of $G(x)$ and the presence of a 0-phase eigenvector of $U(P, x)$ in $\text{row}(A)$, proven in [8, Lemma 32].

▶ **Lemma 24.** *$G(x)$ is not connected if and only if there exists $|\psi\rangle \in \text{row}(A)$ that is fixed by $U(P, x)$. Moreover, if $G(x)$ has $\kappa > 1$ components, there exists a $(\kappa - 1)$-dimensional subspace of* $\text{row}(A)$ *that is fixed by $U(P, x)$.*

Thus, to determine if $G(x)$ is connected, it is sufficient to detect the presence of *any* 0-phase eigenvector of $U(P, x)$ on $\text{row}(A)$. Let $\{|\psi_i\rangle\}_{i=1}^{n-1}$ be any basis for $\text{row}(A)$, not necessarily orthogonal, and suppose we have access to an operation that generates $|\psi_{\text{init}}\rangle = \sum_{i=1}^{n-1} |i\rangle|\psi_i\rangle$. Such a basis is independent of the input, so we can certainly perform such a map with 0 queries. We will later discuss cases in which we can implement such a map time efficiently.

▶ **Algorithm 25.** *Assume there is a known constant λ such that if $G(x)$ is connected, then $\lambda_2(G(x)) \geq \lambda$. Let $\{|\psi_i\rangle\}_i$ be some states that span the rowspace of A, whose choice determines the cost of the amplitude estimation step.*

1. *Prepare $|\psi_{\text{init}}\rangle = \sum_{i=1}^{n-1} \frac{1}{\sqrt{n-1}}|i\rangle|\psi_i\rangle$.*
2. *Perform the phase estimation procedure of [8, Theorem 9] of $U(P,x)$ on the second register, to precision $\sqrt{\lambda/d_{\max}(G)}$, and accuracy ϵ.*
3. *Use amplitude estimation to determine if the amplitude on $|0\rangle$ in the phase register is 0, in which case, output "connected", or > 0, in which case, output "not connected."*

The algorithm performs phase estimation (see [8, Theorem 8]) on the second register of $|\psi_{\text{init}}\rangle$, with precision $\sqrt{\lambda/d_{\max}(G)}$. Intuitively, this will distinguish any part of the second register that is in the 0-phase space of $U(P,x)$, labeling it with $|0\rangle$ in a new phase register, from any part of the state that is in the span of the θ-phase vectors of $U(P,x)$ for $|\theta| > \sqrt{\lambda/d_{\max}(G)}$. We can then estimate the part of the state in the 0-phase space of $U(P,x)$ by using amplitude estimation in Step 3. First, suppose that there are $\kappa - 1 > 0$ orthonormal 0-phase eigenvectors of $U(P,x)$ in $\text{row}(A)$, and let Π be the orthonormal projector onto their span. By [8, Theorem 8], for each i, the phase estimation step will map $|i\rangle\,(\Pi|\psi_i\rangle)$ to $|i\rangle|0\rangle\,(\Pi|\psi_i\rangle)$. Thus, the squared amplitude on $|0\rangle$ in the phase register will be at least:

$$\varepsilon := \frac{\|(I\otimes\Pi)|\psi_{\text{init}}\rangle\|^2}{\||\psi_{\text{init}}\rangle\|^2} = \frac{1}{\||\psi_{\text{init}}\rangle\|^2}\sum_{i=1}^{n-1}\|\Pi|\psi_i\rangle\|^2 > 0, \tag{6}$$

since the $|\psi_i\rangle$ span $\text{row}(A)$.

On the other hand, suppose $G(x)$ is connected, so there is no 0-phase eigenvector in $\text{row}(A)$. Then all phases will be at least $\Delta(U(P,x)) \geq \sqrt{\lambda/d_{\max}(G)}$, so the phase register will have squared overlap at most ϵ with $|0\rangle$. Setting $\epsilon = \varepsilon/2$, we just need to distinguish between an amplitude of $\geq \varepsilon$ and an amplitude of $\leq \varepsilon/2$ on $|0\rangle$, which we can do using amplitude estimation in $\frac{1}{\sqrt{\varepsilon}}$ calls to Steps 1 and 2 (See [8] for details). Step 2 can be implemented using $\sqrt{d_{\max}(G)/\lambda}\log\frac{1}{\varepsilon}$ calls to $U(P,x)$. By [9, Theorem 13], if U is the cost of implementing, for any $u \in V$, the map

$$|u,0\rangle \mapsto \sum_{(v,\ell)\in\Gamma(u)} \sqrt{c(u,v,\ell)/d_G(u)}|u,v,\ell\rangle, \tag{7}$$

which corresponds to one step of a quantum walk on G, then $U(P,x)$ can be implemented in time $O(\mathsf{U})$. We thus get the following (formally proven in [8, Theorem 34]):

▶ **Theorem 26.** *Fix $\lambda > 0$. Let Init denote the cost of generating the initial state $|\psi_{\text{init}}\rangle$, and U the cost of the quantum walk step in equation 7. Let ε be as in equation 6. Then for any family of connected graphs G and $X \subseteq \{0,1\}^{E(G)}$ such that for all $x \in X$, either $\lambda_2(G(x)) \geq \lambda$ or $G(x)$ is not connected, $\text{CONN}_{G,X}$ can be decided by a quantum algorithm with cost $O\left(\frac{1}{\sqrt{\varepsilon}}\left(\mathsf{Init} + \sqrt{\frac{d_{\max}(G)}{\lambda}}\mathsf{U}\log\frac{1}{\varepsilon}\right)\right)$.*

In [8, Sections 5.1 and 5.2], we discuss particular implementations of this algorithm, but if we only care about query complexity, we already have Corollary 19. We formally prove Corollary 19 in [8], but the idea is that Init costs 0 queries, $U(P,x)$ can be implemented in 2 queries, and if there are $\kappa - 1$ orthonormal 0-phase vectors of $U(P,x)$ in $\text{row}(A)$, they each contribute at least $\frac{1}{n-1}$ to $\epsilon = \|(I\otimes\Pi)|\psi_{\text{init}}\rangle\|^2$, so $\epsilon \geq \frac{\kappa-1}{n-1}$, giving a total query complexity of $O(\sqrt{nd_{\max}(G)(\kappa\lambda)^{-1}})$.

4.1 Estimating the connectivity when G is a complete graph

For the remainder of this section, let G be the complete graph on n vertices, K_n. In that case, we can not only decide if $G(x)$ is connected, but estimate $\lambda_2(G(x))$. The idea is to relate the smallest phase of $U(P,x)$ on $\mathrm{row}(A)$ to $\lambda_2(G(x))$, and estimate this value using quantum phase estimation. We use a correspondence between the phases of the product of two reflections $U = (2\Pi_A - I)(2\Pi_B - 1)$ and the singular values of its discriminant, defined $\Pi_A\Pi_B$ due to Szegedy [16] to prove the following in [8, Section 5.3].

▶ **Lemma 27.** *Let $U = (2\Pi_A - I)(2\Pi_B - I)$ and $D = \Pi_A\Pi_B$ be its discriminant. Then $\Delta(-U) = 2\sin^{-1}(\sigma_{\min}(D))$. Moreover, when G is a complete graph on n vertices, we have for any x, $\lambda_2(G(x)) = n\sin^2(\Delta(U(P,x))/2)$.*

This correspondence also implies the following, which allows us to restrict our attention to $\mathrm{row}(A)$ in searching for the smallest phase of $U(P,x)$ and is proven in [8, Section 5.3]:

▶ **Lemma 28.** *Let $U = (2\Pi_A - I)(2\Pi_B - I)$, and let $|\Delta_+\rangle$ be a $\Delta(U)$-eigenvector of U, and $|\Delta_-\rangle$ a $(-\Delta(U))$-eigenvector of U. Then there exists a vector $|u\rangle$ in the support of A such that $|u\rangle \in \mathrm{span}\{|\Delta_+\rangle, |\Delta_-\rangle\}$. In particular, if $|\Delta_\pm\rangle$ are $\pm\Delta(U(P,x))$-eigenvectors of $U(P,x)$, then there exists a vector $|u\rangle$ in $\mathrm{row}(A)$ such that $|u\rangle \in \mathrm{span}\{|\Delta_+\rangle, |\Delta_-\rangle\}$.*

We will estimate the value $\tau = \Delta(U(P,x))/\pi$ in the range $[0,1]$, which we will then transform into an estimate of $\lambda_2(G(x))$. At every iteration, c will denote a lower bound for τ and C will denote the current upper bound. At the beginning of the algorithm we have $c = 0$, $C = 1$, and every iteration will result in updating either C or c in such a manner that the new interval for τ is reduce by a fraction of $2/3$. The algorithm is described in Algorithm 29.

▶ **Algorithm 29.** *To begin, let $c = 0$ and $C = 1$.*

```
1. Set $\varphi = \frac{C-c}{3}$, $\epsilon = \frac{1}{\sqrt{2n}}$, $\delta = c + \varphi$.
2. For $j = 1, \ldots, 4\log(n/\varepsilon)$:
   a. Prepare $\sum_{i=1}^{n-1} \frac{1}{\sqrt{n-1}}|i\rangle|\psi_i\rangle|0\rangle_B|0\rangle_P$.
   b. Perform the gapped phase estimation algorithm $GPE(\varphi,\epsilon,\delta)$ of [8, Theorem 9] to
      $U(P,x)$ on the second register.
   c. Use amplitude estimation to distinguish between the case when the amplitude on $|0\rangle_B$
      is $\geq \frac{1}{\sqrt{n}}$, in which case output "$a_j = 0$", and the case where the ampltiude is $\leq \frac{1}{\sqrt{2n}}$,
      in which case, output "$a_j = 1$".
3. Compute $\tilde{a} = Maj(a_1, \ldots, a_{4\log(n/\varepsilon)})$. If the result is $0$, set $C = \delta + \varphi$. If the result is $1$,
   set $c = \delta$. If $C - c \leq 2\varepsilon c$, then output $n\sin^2\left(\frac{\pi(C+c)}{4}\right)$. Otherwise, return to Step 1.
```

We say an iteration of the algorithm *succeeds* if $\tilde{a} = Maj(a_1, \ldots, a_{4\log(n/\varepsilon)})$ correctly indicates whether the amplitude on $|0\rangle_B$ is $\geq n^{-1/2}$ or $\leq (2n)^{-1/2}$. This happens with probability $\Omega(1-(\varepsilon/n)^4)$. Since we will shortly see that the algorithm runs for at most $\widetilde{O}(n/\varepsilon\sqrt{\lambda_2(G(x))}) \leq \widetilde{O}\left(n^2\varepsilon^{-1}\right)$ steps, a Taylor series approximation guarantees that the probability that every iteration succeeds is at least $\Omega\left(1-(\varepsilon/n)^2\right)$. It is therefore reasonable to assume that every iteration succeeds, since this happens with high probability. We first note that if every iteration succeeds, throughout the algorithm we have $\tau = \Delta(U(P,x))/\pi \in [c,C]$.

▶ **Lemma 30.** *Let $\tau = \Delta(U(P,x))/\pi$. For any φ and δ, if $\tau \geq \delta + \varphi$, applying $GPE(\varphi,\epsilon,\delta)$ to $|\psi_{\mathrm{init}}\rangle$ results in a state with amplitude at most $\frac{1}{\sqrt{2n}}$ on $|0\rangle_B$ in register B; and if $\tau \leq \delta$, this results in a state with amplitude at least $\frac{1}{\sqrt{n}}$ on $|0\rangle_B$ in register B. Thus, if every iteration succeeds, at every iteration, we have $\tau \in [c,C]$.*

The proof of Lemma 30 is found in [8, Section 5.3]. Next, we analyze the running time of Algorithm 29 to get the following theorem, also proven in [8, Section 5.3].

▶ **Theorem 31.** *With probability $\Omega(1-(\varepsilon/n)^2)$, Algorithm 29 will terminate after time $\widetilde{O}(n/\varepsilon\sqrt{\lambda_2(G(x))})$.*

Finally, we prove in [8, Section 5.3] that the algorithm outputs an estimate that is within ε multiplicative error of $\lambda_2(G(x))$. Theorem 22 follows from Theorems 31 and 32.

▶ **Theorem 32.** *With probability at least $\Omega(1-(\varepsilon/n)^2)$, Algorithm 29 outputs an estimate $\tilde{\lambda}$ such that $\left|\lambda_2(G(x))-\tilde{\lambda}\right| \leq \frac{\pi^2 3}{4}\varepsilon\lambda_2(G(x))$.*

References

1 A. Āriņš. *Span-Program-Based Quantum Algorithms for Graph Bipartiteness and Connectivity*, pages 35–41. Springer International Publishing, Cham, 2016. `doi:10.1007/978-3-319-29817-7_4`.

2 A. Belovs. Learning-graph-based quantum algorithm for k-distinctness. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012)*, pages 207–216, 2012. `doi:10.1109/FOCS.2012.18`.

3 A. Belovs. Span programs for functions with constant-sized 1-certificates. In *Proceedings of the 44th Symposium on Theory of Computing (STOC 2012)*, pages 77–84, 2012.

4 A. Belovs and B. W. Reichardt. Span programs and quantum algorithms for *st*-connectivity and claw detection. In *Proceedings of the 20th European Symposium on Algorithms (ESA 2012)*, pages 193–204, 2012.

5 C. Cade, A. Montanaro, and A. Belovs. Time and space efficient quantum algorithms for detecting cycles and testing bipartiteness, 2016. `arXiv:1610.00581`.

6 C. Dürr, M. Heiligman, P. Høyer, and M. Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006.

7 T. Ito and S. Jeffery. Approximate span programs. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, pages 12:1–12:14, 2016. `arXiv:1507.00432`.

8 Michael Jarrett, Stacey Jeffery, Shelby Kimmel, and Alvaro Piedrafita. Quantum algorithms for connectivity and related problems. *arXiv preprint arXiv:1804.10591*, 2018.

9 S. Jeffery and S. Kimmel. Quantum algorithms for graph connectivity and formula evaluation. *Quantum*, 1:26, 2017. `doi:10.22331/q-2017-08-17-26`.

10 M. Karchmer and A. Wigderson. On span programs. In *Proceedings of the 8th Annual IEEE Conference on Structure in Complexity Theory*, pages 102–111, 1993.

11 N. Nisan and A. Ta-Shma. Symmetric logspace is closed under complement. In *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing (STOC 1995)*, pages 140–146, New York, NY, USA, 1995. ACM. `doi:10.1145/225058.225101`.

12 B. W. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every Boolean function. In *Proceedings of the 50th IEEE Symposium on Foundations of Computer Science (FOCS 2009)*, pages 544–551, 2009. `arXiv:quant-ph/0904.2759`.

13 B. W. Reichardt. Span programs and quantum query algorithms. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:110, 2010.

14 B. W. Reichardt. Reflections for quantum query algorithms. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2011)*, pages 560–569. SIAM, 2011.

15 B. W. Reichardt and R. Špalek. Span-program-based quantum algorithm for evaluating formulas. *Theory of Computing*, 8(13):291–319, 2012.

16 M. Szegedy. Quantum speed-up of Markov chain based algorithms. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2004)*, pages 32–41, Washington, DC, USA, 2004. IEEE Computer Society. `doi:10.1109/FOCS.2004.53`.

Generalized Coloring of Permutations

Vít Jelínek
Computer Science Institute, Charles University
Malostranské náměstí 25, Praha 1, 11800, Czechia
jelinek@iuuk.mff.cuni.cz
https://orcid.org/0000-0003-4831-4079

Michal Opler
Computer Science Institute, Charles University
Malostranské náměstí 25, Praha 1, 11800, Czechia
opler@iuuk.mff.cuni.cz
https://orcid.org/0000-0002-4389-5807

Pavel Valtr
Department of Applied Mathematics, Charles University
Malostranské náměstí 25, Praha 1, 11800, Czechia
https://orcid.org/0000-0002-3102-4166

Abstract

A permutation π is a *merge* of a permutation σ and a permutation τ, if we can color the elements of π red and blue so that the red elements have the same relative order as σ and the blue ones as τ. We consider, for fixed hereditary permutation classes $\mathcal{C}$ and $\mathcal{D}$, the complexity of determining whether a given permutation π is a merge of an element of $\mathcal{C}$ with an element of $\mathcal{D}$.

We develop general algorithmic approaches for identifying polynomially tractable cases of merge recognition. Our tools include a version of nondeterministic logspace streaming recognizability of permutations, which we introduce, and a concept of bounded width decomposition, inspired by the work of Ahal and Rabinovich.

As a consequence of the general results, we can provide nontrivial examples of tractable permutation merges involving commonly studied permutation classes, such as the class of layered permutations, the class of separable permutations, or the class of permutations avoiding a decreasing sequence of a given length.

On the negative side, we obtain a general hardness result which implies, for example, that it is NP-complete to recognize the permutations that can be merged from two subpermutations avoiding the pattern 2413.

2012 ACM Subject Classification Mathematics of computing → Permutations and combinations, Mathematics of computing → Combinatorial algorithms, Theory of computation → Problems, reductions and completeness

Keywords and phrases Permutations, merge, generalized coloring

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.50

Funding Supported by the GAUK project 1766318, by project Impuls of the Neuron Fund for Support of Science, and by project 18-19158S of the Czech Science Foundation.

© Vít Jelínek, Michal Opler, and Pavel Valtr;
licensed under Creative Commons License CC-BY

1 Introduction

Definitions and previous results

A *permutation* is a sequence $\pi = \pi_1, \pi_2, \ldots, \pi_n$ in which each number from the set $[n] = \{1, 2, \ldots, n\}$ appears exactly once. We then say that a permutation $\pi = \pi_1, \ldots, \pi_n$ *contains* a permutation $\sigma = \sigma_1, \ldots, \sigma_k$, if π has a subsequence of length k whose elements have the same relative order as the elements of σ (see Section 2 for a more formal definition). If π does not contain σ, we say that π *avoids* σ, or π is σ*-avoiding*.

A permutation π is a *merge* of a permutation σ and a permutation τ, if we can color the elements of π with colors red and blue so that the red elements have the same relative order as σ and the blue ones as τ. For two sets of permutations $\mathcal{C}$ and $\mathcal{D}$, we let $\mathcal{C} \odot \mathcal{D}$ denote the set of the permutations that can be obtained by merging a permutation $\tau \in \mathcal{C}$ with a permutation $\sigma \in \mathcal{D}$.

In this paper, we study the algorithmic complexity of determining whether a given permutation is a merge of a pair of permutations with a prescribed structure. More formally, for a fixed pair of hereditary permutation classes $\mathcal{C}$ and $\mathcal{D}$, we consider the complexity of determining whether a given permutation π belongs to $\mathcal{C} \odot \mathcal{D}$.

The notion of merge has been originally introduced as an approach for the enumeration of pattern-avoiding permutations [5, 4]. For instance, Claesson et al. [13] have shown that every 1324-avoiding permutation can be obtained by merging a 132-avoiding permutation with a 213-avoiding one, and this result, and its subsequent strengthenings by Bóna [8, 9] and Bevan et al. [7], are the basis of the best known upper bounds for the number of 1324-avoiding permutations.

Apart from enumeration questions, the research into permutation merges has also addressed structural issues, such as whether a given permutation class can be obtained by merging two of its proper subclasses [18, 17], or which classes can be obtained by merging a bounded number of permutations from a given class [3, 19, 21]. Our paper is, however, the first to address algorithmic aspects of permutation merges.

So far, most of the algorithmic research related to permutations has focused on the decision problem known as *Permutation Pattern Matching*, or PPM, where the goal is to determine whether a given permutation π (the 'pattern') is contained in a permutation τ (the 'text'). Bose et al. [11] have shown that PPM is NP-complete for general π and τ, but it is polynomial when π is restricted to the class of the so-called separable permutations. The latter result was generalized by Ahal and Rabinovich [2]. More precisely, Ahal and Rabinovich introduced a notion of tree decomposition for permutations and an associated width parameter, closely related to the concept of tree-width from graph theory; they then proved that PPM is polynomial when the pattern π is restricted to a class of bounded tree-width, of which separable permutations are a special case. We remark that a different width parameter for permutations was introduced by Guillemot and Marx [16], who used it to prove that PPM is in FPT with the length of the pattern π as the parameter. While the results on the complexity of PPM do not have any immediate consequences for the problems we consider in this paper, the tree-width concept of Ahal and Rabinovich is a crucial ingredient in our results.

The decision problem of recognizing permutations from $\mathcal{C} \odot \mathcal{D}$ can be viewed as a permutation analogue of the generalized graph coloring problem from graph theory. For a fixed k-tuple $\mathcal{G}_1, \ldots, \mathcal{G}_k$ of graph classes, a generalized coloring of a graph is an assignment of colors $1, 2, \ldots, k$ to its vertices so that the vertices of color i induce a subgraph from $\mathcal{G}_i$. In particular, if all the $\mathcal{G}_i$ are equal to the class of edgeless graphs, this notion reduces to the classical notion of k-coloring. The research into the complexity of generalized graph coloring was initiated by Rutenburg [20], who considered graph properties defined by a finite set of forbidden subgraphs. Later, Farrugia [15] has shown that if all the $\mathcal{G}_i$ are hereditary and additive (i.e., closed under taking induced subgraphs and forming disjoint unions) then the problem is NP-hard, except the trivially polynomial case when $k = 2$ and both $\mathcal{G}_1$ and $\mathcal{G}_2$ are equal to the class of edgeless graphs. Further results in this area were obtained, e.g., by Brown [12], Alexeev et al. [6], Achlioptas et al. [1], or Borowiecki [10].

As with generalized graph coloring, the recognition of permutation merges admits several cases which are trivially polynomial. For instance, let $\mathcal{I}_k$ be the class of permutations that can be merged from at most k increasing subsequences, or equivalently, of permutations that avoid the pattern $k+1, k, \dots, 1$. Similarly, let $\mathcal{D}_k$ be the permutations merged from at most k decreasing subsequences, which are exactly the avoiders of $1, 2, \dots, k+1$. One may easily see that $\mathcal{I}_k \odot \mathcal{I}_\ell = \mathcal{I}_{k+\ell}$ and $\mathcal{D}_k \odot \mathcal{D}_\ell = \mathcal{D}_{k+\ell}$, and in particular, these merges are trivially polynomially recognizable. Moreover, Kézdy et al. [19] have shown that for any $k, \ell \geq 1$, the class $\mathcal{I}_k \odot \mathcal{D}_\ell$ has only finitely many minimal excluded patterns, and therefore these classes are polynomially recognizable as well.

Ekim et al. [14] studied the complexity of generalized 2-colorings when the input graph is restricted to the class of the so-called permutation graphs. Their results, in our terminology, imply the polynomial recognition of $\mathcal{L} \odot \mathcal{I}_1$ and of $\mathcal{L} \odot \overline{\mathcal{L}}$, where $\mathcal{L}$ and $\overline{\mathcal{L}}$ denote the classes of layered and co-layered permutations (see Section 2 for definitions).

Our results

In this paper, we show that there are many more cases of polynomially tractable merges of permutation classes. This contrasts with Farrugia's above-mentioned result on generalized graph coloring. As our main results, we will present two general approaches to show that a permutation class of the form $\mathcal{C} \odot \mathcal{D}$ is polynomially recognizable.

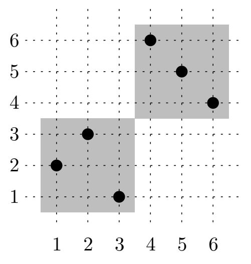
Our first approach, which we present in Section 3.1, is based on the concept of non-deterministically logspace on-line recognizable (or NLOL-recognizable) permutation classes, which we introduce. We will show that an arbitrary merge of NLOL-recognizable classes is polynomially recognizable. While this approach is conceptually quite simple, it generalizes all the previously known examples of tractable merges following from the work of Kézdy et al. [19] and Ekim et al. [14].

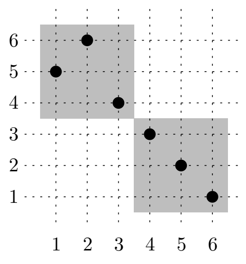
For our second approach, presented in Section 4, we introduce the notion of grid decompositions, and the associated width parameter called grid-width. We combine the grid decomposition technique with a restricted version of NLOL, called 2D-NLOL, to prove that the merge of a 2D-NLOL-recognizable class with a class of bounded grid-width can be recognized in polynomial time. This approach allows us to handle further cases of natural permutation classes that are not NLOL-recognizable, such as the class of 213-avoiders, or the class of separable permutations.

To complement our tractability results, we also provide, in Section 5, an NP-hardness result. The result implies, among other examples, that the recognition of $\mathrm{Av}(2413) \odot \mathrm{Av}(2413)$ is NP-hard, where $\mathrm{Av}(2413)$ is the class of 2413-avoiding permutations.

2 Basic definitions

A *permutation of order* n is a sequence in which each element of the set $[n]$ appears exactly once. We let $\mathcal{S}_n$ denote the set of permutations of order n. When writing out short permutations explicitly, we shall omit all punctuation and write, e.g., 15342 for the


$231 \oplus 321 = 231654$


$231 \ominus 321 = 564321$

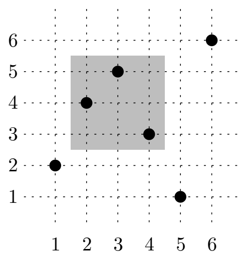

the '3' of 2314 inflated by 231

Figure 1 Example of direct sum (left), skew sum (center) and inflation (right).

permutation $1, 5, 3, 4, 2 \in \mathcal{S}_5$. We shall assume that there is a unique permutation of order 0, corresponding to the empty sequence.

To represent a permutation $\pi = \pi_1, \pi_2, \dots, \pi_n$ graphically, we will use the *permutation diagram*, which is the set of points $\{(i, \pi_i);\ i \in [n]\}$ in the plane. See Figure 1 for an example. Note that we use Cartesian coordinates, that is, the first row of the diagram is at the bottom.

Let $x = x_1, x_2, \dots, x_n$ and $y = y_1, y_2, \dots, y_n$ be two sequences of numbers. We say that x and y are *order-isomorphic*, if, for every $1 \le i, j \le n$ we have $x_i < x_j \iff y_i < y_j$. A permutation $\pi \in \mathcal{S}_n$ *contains* a permutation $\sigma \in \mathcal{S}_k$, if π has a subsequence order-isomorphic to σ. Such a subsequence is then an *occurrence* (or a *copy*) of σ in π. If π does not contain σ, we say that π *avoids* σ.

A *hereditary permutation class* (or just *permutation class*, for short) is a set $\mathcal{C}$ of permutations with the property that if π is in $\mathcal{S}$, then all the permutations contained in π are in $\mathcal{C}$ as well. For a permutation σ, we let $\mathrm{Av}(\sigma)$ denote the set of σ-avoiding permutations. More generally, for a set F of permutations, we let $\mathrm{Av}(F)$ be the set of permutations that avoid all the elements of F. Clearly, $\mathrm{Av}(F)$ is a permutation class, and any permutation class is equal to $\mathrm{Av}(F)$ for a (possibly infinite) set F.

Consider a pair of permutations $\sigma = \sigma_1, \dots, \sigma_k \in \mathcal{S}_k$ and $\tau = \tau_1, \dots, \tau_\ell \in \mathcal{S}_\ell$. The *direct sum* of σ and τ, denoted $\sigma \oplus \tau$, is the permutation $\pi = \sigma_1, \dots, \sigma_k, k + \tau_1, k + \tau_2, \dots, k + \tau_\ell \in \mathcal{S}_{k+\ell}$. Similarly, their *skew sum*, denoted $\sigma \ominus \pi$, is the permutation $\ell + \sigma_1, \dots, \ell + \sigma_k, \tau_1, \tau_2, \dots, \tau_\ell \in \mathcal{S}_{k+\ell}$; see Figure 1.

For a pair of permutation classes $\mathcal{C}$ and $\mathcal{D}$, we let $\mathcal{C} \oplus \mathcal{D}$ be the set $\{\sigma \oplus \tau;\ \sigma \in \mathcal{C}, \tau \in \mathcal{D}\}$; note that this is again a permutation class. The class $\mathcal{C} \ominus \mathcal{D}$ is defined analogously. The *sum-closure* of a class $\mathcal{C}$, denoted $\mathcal{C}^{\oplus}$, is the class of all the permutations that can be obtained as a direct sum of finitely many members of $\mathcal{C}$; the *skew-closure* $\mathcal{C}^{\ominus}$ is defined analogously.

A permutation π is a *merge* of permutations σ and τ if we can color the elements of π with colors red and blue so that the red elements are order-isomorphic to σ and the blue ones to τ. The merge of a class $\mathcal{C}$ and a class $\mathcal{D}$ is the class $\mathcal{C} \odot \mathcal{D}$ of permutations that can be obtained by merging an element of $\mathcal{C}$ with an element of $\mathcal{D}$.

For integers i and j, we let $[i, j]$ denote the set $\{k \in \mathbb{Z};\ i \le k \le j\}$. A set of this form is an *integer interval.* We also use the notation $[i, j)$ for the interval $[i, j-1]$ and $(i, j]$ for $[i+1, j]$. A *box* is the Cartesian product of two integer intervals. For a box $B = I \times J \subseteq [n] \times [n]$ and a permutation $\pi = \pi_1, \dots, \pi_n$, the *restriction* of π to B, denoted $\pi|_B$, is the subsequence of π formed by the entries π_i satisfying $i \in I$ and $\pi_i \in J$.

Let $\sigma = \sigma_1, \dots, \sigma_k$ and $\tau = \tau_1, \dots, \tau_\ell$ be again a pair of nonempty permutations. The *inflation* of an element σ_i of σ by τ, is an operation which produces a permutation

$$\pi = \sigma'_1, \sigma'_2, \dots, \sigma'_{i-1}, \tau'_1, \tau'_2, \dots, \tau'_\ell, \sigma'_{i+1}, \dots, \sigma'_k,$$

where the subsequence $\tau'_1, \tau'_2, \ldots, \tau'_\ell$ is a copy of τ, and for any $j \in \ell$, the subsequence $\sigma'_1, \sigma'_2, \ldots, \sigma'_{i-1}, \tau'_j, \sigma'_{i+1}, \ldots, \sigma'_k$ is a copy of σ; see again Figure 1. A permutation is *simple*, if it cannot be obtained from two strictly smaller permutations by an inflation. For instance, the permutation 25314 is simple, while 25341 is not, since it can be obtained, e.g., by inflating the element '3' in 2431 by the permutation 12.

For a permutation $\pi = \pi_1, \ldots, \pi_n$ the *reverse* of π is the permutation $\pi_n, \pi_{n-1}, \ldots, \pi_1$, the *complement* of π is the permutation $n+1-\pi_1, n+1-\pi_2, \ldots, n+1-\pi_n$, and the *inverse* of π is the permutation $\sigma = \sigma_1, \ldots, \sigma_n$ satisfying $\pi_i = j \iff \sigma_j = i$. We let π^r, π^c and π^{-1} denote the reverse, complement and inverse of π, respectively. Similarly, for a class of permutations $\mathcal{C}$, we let $\mathcal{C}^r$ denote the set $\{\pi^r;\ \pi \in \mathcal{C}\}$, and similarly for $\mathcal{C}^c$ and $\mathcal{C}^{-1}$. Note that $\mathcal{C}^r$, $\mathcal{C}^c$ and $\mathcal{C}^{-1}$ are again permutation classes.

Several commonly encountered permutation classes have standard names in the literature. The *increasing permutations* are the permutations from the class $\mathrm{Av}(21)$ and symmetrically, the elements of $\mathrm{Av}(12)$ are the *decreasing permutations*. The permutations avoiding both 231 and 312 are known as the *layered permutations*. Layered permutations can also be characterized as those permutations that can be written as a finite direct sum in which each summand is a decreasing permutation; that is, the class of layered permutations is the sum-closure of $\mathrm{Av}(12)$. The complements of layered permutations are known as the *co-layered permutations*; they form the class $\mathrm{Av}(\{132, 213\})$. Finally, the permutations from the class $\mathrm{Av}(\{2413, 3142\})$ are known as the *separable permutations*; it is known [11] that these are precisely the permutations that can be created from the permutation of size 1 by direct sums and skew sums.

3 Tractable merges

For a permutation class $\mathcal{C}$, *$\mathcal{C}$-recognition* is the decision problem to determine whether a given permutation belongs to $\mathcal{C}$. Our main goal is to identify pairs of classes $\mathcal{C}, \mathcal{D}$ for which the $(\mathcal{C} \odot \mathcal{D})$-recognition problem is tractable, i.e., solvable in polynomial time.

3.1 NLOL-recognizable classes

Our first nontrivial example of classes whose merges can be efficiently recognized are the so-called NLOL-recognizable permutation classes. Informally speaking, a permutation class is NLOL-recognizable if its members can be recognized by a single-pass nondeterministic streaming algorithm with logarithmic memory.

More formally, we say that a permutation class $\mathcal{C}$ is *nondeterministically logspace on-line* recognizable, or NLOL-recognizable for short, if there is a nondeterministic algorithm A that recognizes $\mathcal{C}$ in the following setting: as the first part of the input, the algorithm A receives a number n, which is an upper bound on the length and also on the largest value in the input sequence. The algorithm is then given access to $O(\log n)$ bits of memory, and it receives a sequence of distinct values $\pi_1, \ldots, \pi_k$ from the set $[n]$, terminated by a special symbol `EOF`. Upon receiving the `EOF` symbol, A answers whether the input sequence is order-isomorphic to a permutation in $\mathcal{C}$. The algorithm can store arbitrary data of size $O(\log n)$ in its memory, but as soon as it reads the input value π_i, it can no longer access the previous values of the input. A is nondeterministically recognizing $\mathcal{C}$ in the sense that the input sequence is order-isomorphic to a permutation in $\mathcal{C}$ if and only if at least one computation of A accepts it. The algorithm A is then called an NLOL-recognizer of $\mathcal{C}$. Note that the input sequence is guaranteed to consist of distinct values, so the NLOL-recognizer itself does not need to verify this property. This also implies that the input sequence has length at most n.

We let NLOL denote the set of the NLOL-recognizable permutation classes. Clearly, for any permutation class $\mathcal{C} \in$ NLOL, the $\mathcal{C}$-recognition problem is tractable, since nondeterministic logspace computations can be simulated in polynomial time.

One may easily observe that NLOL contains any finite permutation class, as well as the classes Av(12) and Av(21). The key feature of NLOL is that it is closed under many important operations with permutation classes, including the merge operation.

▶ **Lemma 1.** *If $\mathcal{C}$ and $\mathcal{D}$ are NLOL-recognizable classes, then the following classes are NLOL-recognizable as well:*
(a) *The classes $\mathcal{C} \cap \mathcal{D}$ and $\mathcal{C} \cup \mathcal{D}$.*
(b) *The classes $\mathcal{C}^r$ and $\mathcal{C}^c$.*
(c) *The classes $\mathcal{C}^\oplus$ and $\mathcal{C}^\ominus$, i.e., the sum-closure and skew-closure of $\mathcal{C}$.*
(d) *The classes $\mathcal{C} \oplus \mathcal{D}$ and $\mathcal{C} \ominus \mathcal{D}$.*
(e) *The class $\mathcal{C} \odot \mathcal{D}$.*

▶ **Corollary 2.** *For any sequence of classes $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k \in$ NLOL, the class $\mathcal{C}_1 \odot \mathcal{C}_2 \odot \cdots \odot \mathcal{C}_k$ is in NLOL, and therefore polynomially recognizable.*

Lemma 1 shows that NLOL contains many important permutation classes, including the classes of layered and co-layered permutations, as well as any class of the form $\mathrm{Av}(1, 2, 3 \dots, k)$ or $\mathrm{Av}(k, k-1, \dots, 1)$.

On the negative side, it can be shown that NLOL does not contain some other important classes, such as the class of separable permutations, or its subclasses Av(231), Av(213), Av(312) and Av(132). These five classes share a common feature: their elements have a simple recursive tree-like structure involving direct sums, skew sums and inflations. We shall soon formalize this notion of tree-like structure via the concept of bounded grid-width, and show that it leads to another general type of tractable merges. Before we get there, however, we first introduce a restricted form of NLOL that will play an important part in conjunction with bounded grid-width classes.

3.2 2D-NLOL-recognizable classes

Informally speaking, a permutation class is 2D-NLOL-recognizable, if its members can be recognized by a single-pass nondeterministic streaming algorithm over a sequence of index-value pairs in a left-to-right, bottom-to-top order.

Let $P = \{(x_1, y_1), \dots, (x_n, y_n)\}$ be a set of points in the plane. We say that P is *in general position* if no two of its points are on the same horizontal or vertical line, i.e., there is no $i \neq j$ with $x_i = x_j$ or $y_i = y_j$. We say that a permutation $\pi \in \mathcal{S}_n$ is *shape-isomorphic* to P if there is a bijection $f\colon [n] \to [n]$ such that for every i and j the following holds: $i < j$ if and only if $x_{f(i)} < x_{f(j)}$ and $\pi_i < \pi_j$ if and only if $y_{f(i)} < y_{f(j)}$. Note that a permutation $\pi \in \mathcal{S}_n$ is shape-isomorphic to its diagram $\{(i, \pi_i);\ i \in [n]\}$.

Let $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)$ be a sequence of distinct points in general position. We say that the sequence is *top-right monotone* if for every $i \in [k]$ the point (x_i, y_i) is to the right or above all the previous points of the sequence; formally, for every $i \in [k]$, either for every $j < i$ we have $x_j < x_i$ or for every $j < i$ we have $y_j < y_i$. Note that there can be several top-right monotone sequences corresponding to a single point set. Note also, that a sequence $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k) \subseteq [n] \times [n]$ in general position is top-right monotone if and only if for every $i \in [k]$ there is a box $B_i = [1, r_i] \times [1, t_i]$ which contains the points $(x_1, y_1), \dots, (x_i, y_i)$ but none of the points $(x_{i+1}, y_{i+1}), \dots, (x_k, y_k)$. A sequence of points is *admissible* if it is in general position and top-right monotone.

We will now consider nondeterministic logspace algorithms that receive an integer n, followed by an admissible sequence of points in $[n] \times [n]$ as their input. To describe the assumptions we make about the algorithms, we first introduce some terminology. Let A be such nondeterministic algorithm. A *position* of the algorithm A is a pair (p, m), where $p = (p_x, p_y)$ is a point in $[n+1] \times [n+1]$ and m is a memory state of A. Let S be an admissible sequence of points. We say that A *can reach the position* (p, m) *on input* S, if S is contained in the box $[1, p_x) \times [1, p_y)$ and there is a computation of A starting from its initial state and ending in state m after processing S. Let $p = (p_x, p_y)$ and $p' = (p'_x, p'_y)$ be two points with $1 \le p_x \le p'_x \le n$ and $1 \le p_y \le p'_y \le n$, and S be an admissible sequence. We say that A can *reach position* (p', m') *from position* (p, m) *on input* S, if S is contained in the box $[1, p'_x) \times [1, p'_y)$ but disjoint from the box $[1, p_x) \times [1, p_y)$, and the algorithm A has a computation starting in state m and ending in state m' after processing S.

We say that an algorithm A is *order-oblivious* if it has the following property: for any pair of positions (p, m) and (p', m') with $p \le p'$, and for any pair of admissible sequences S and S' that correspond to two top-right monotone orderings of the same point set, A can reach (p', m') from (p, m) on input S if and only if it can reach (p', m') from (p, m) on input S'. Informally speaking, the state reached by an order-oblivious algorithm only depends on the set of points it has received as input, but not on their ordering. We may therefore say, e.g., that A reaches position (p, m) on a set of points P, without specifying the particular ordering of P, with the assumption that the ordering is top-right monotone.

We say that an order-oblivious algorithm A is *box-coherent* if it has the following property: for any indices $i \le i'$ and $j \le j'$, consider the four points $p^{\llcorner} = (i, j)$, $p^{\ulcorner} = (i, j')$, $p^{\lrcorner} = (i', j)$ and $p^{\urcorner} = (i', j')$ and four corresponding memory states $m^{\llcorner}$, $m^{\ulcorner}$, $m^{\lrcorner}$ and $m^{\urcorner}$. Suppose that A can reach the position $(p^{\lrcorner}, m^{\lrcorner})$ from $(p^{\llcorner}, m^{\llcorner})$ on an input $X \subseteq [i, i') \times [1, j)$, and that it can reach $(p^{\ulcorner}, m^{\ulcorner})$ from $(p^{\llcorner}, m^{\llcorner})$ on an input $Y \subseteq [1, i) \times [j, j')$. Let Z be a subset of $[i, i') \times [j, j')$ such that $X \cup Y \cup Z$ is in general position. Let $(p^{\urcorner}, m^{\urcorner})$ be a position reachable from $(p^{\lrcorner}, m^{\lrcorner})$ on input $Y \cup Z$. Then the reachability of $(p^{\urcorner}, m^{\urcorner})$ from $(p^{\ulcorner}, m^{\ulcorner})$ on input $X \cup Z$ only depends on the four states $m^{\llcorner}, m^{\lrcorner}, m^{\ulcorner}, m^{\urcorner}$ and the set Z; in particular, it does not depend on the the set X itself. Symmetrically, if we let $(p^{\urcorner}, m^{\urcorner})$ be a position reachable from $(p^{\ulcorner}, m^{\ulcorner})$ on input $X \cup Z$ then the reachability of $(p^{\urcorner}, m^{\urcorner})$ from $(p^{\lrcorner}, m^{\lrcorner})$ on input $Y \cup Z$ only depends on the four memory states and the set Z, but does not depend on Y. Informally, box-coherence means that the memory states $m^{\llcorner}$ and $m^{\lrcorner}$ retain enough information about X to determine the reachable states on inputs of the form $X \cup Z$.

We say that a permutation class $\mathcal{C}$ is *2D nondeterministically logspace on-line* recognizable, or 2D-NLOL-recognizable for short, if there is a nondeterministic order-oblivious box-coherent algorithm A that recognizes $\mathcal{C}$ in the following setting: as the first part of the input, the algorithm A receives a number n, which is an upper bound on the largest value in the input sequence. The algorithm is then given access to $O(\log n)$ bits of memory, and it receives a top-right monotone sequence of points $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ from $[n] \times [n]$, terminated by a special symbol `EOF`. Upon receiving the `EOF` symbol, the algorithm answers whether the input sequence is shape-isomorphic to a permutation in $\mathcal{C}$. The algorithm can store arbitrary data of size $O(\log n)$ in its memory, but after it reads the value (x_i, y_i) from the input, it cannot access any of the previous values. A is nondeterministically recognizing $\mathcal{C}$ in the sense that the input sequence is shape-isomorphic to a permutation in $\mathcal{C}$ if and only if at least one computation of A accepts it. The algorithm A is then called a 2D-NLOL-recognizer of $\mathcal{C}$. Note that the algorithm A does not have to verify that the input is in general position; in other words, on inputs that fail this condition, the behavior of A can be arbitrary.

We let 2D-NLOL denote the set of the 2D-NLOL-recognizable permutation classes. Clearly, 2D-NLOL is contained in NLOL: the left-to-right ordering is a special case of a top-right monotone ordering, and any 2D-NLOL-recognizer can be trivially transformed into an NLOL-recognizer. Furthermore, we observe that 2D-NLOL contains any finite permutation class, as well as the classes Av(12) and Av(21). And like NLOL, 2D-NLOL is closed under many important operations with permutation classes, including the merge operation.

▶ **Lemma 3.** *If $\mathcal{C}$ and $\mathcal{D}$ are 2D-NLOL-recognizable classes, then the following classes are 2D-NLOL-recognizable as well:*

(a) *The classes $\mathcal{C} \cap \mathcal{D}$ and $\mathcal{C} \cup \mathcal{D}$.*
(b) *The class $\mathcal{C}^{-1}$, which contains the inverses of the permutations of $\mathcal{C}$.*
(c) *The classes $\mathcal{C}^{\oplus}$ and $\mathcal{C}^{\ominus}$, i.e., the sum-closure and skew-closure of $\mathcal{C}$.*
(d) *The classes $\mathcal{C} \oplus \mathcal{D}$ and $\mathcal{C} \ominus \mathcal{D}$.*
(e) *The class $\mathcal{C} \odot \mathcal{D}$.*

4 Grid-width

Let us introduce a decomposition of permutations and a corresponding width parameter, which are suited for describing various algorithms using dynamic programming.

An *interval family* I is a set of pairwise disjoint integer intervals with the natural ordering $I_1, \dots, I_n$ such that for $j < k$, $I_j < I_k$. For two interval families $\mathcal{I}$ and $\mathcal{J}$, let $\mathcal{I} \times \mathcal{J}$ denote the naturally defined set of boxes in the plane. For a point set A in the plane, let $x(A)$ denote its projection on the x-axis and equivalently $y(A)$ its projection on the y-axis. The *intervalicity* of a set $A \subseteq [n]$, denoted by $I(A)$, is the size of the smallest interval family whose union is equal to A.

A *grid tree* of a permutation $\pi \in \mathcal{S}_n$ is a rooted binary tree T with n leaves, each leaf being labeled by a distinct point of the permutation diagram $\{(i, \pi_i);\ i \in [n]\}$. Let S_v^T denote the point set of the labels on the leaves in the subtree of T rooted in v. The *grid-width* of a vertex v in T is the maximum of the intervalicities $I(x(S_v^T))$ and $I(y(S_v^T))$, and the *grid-with* of T, denoted by $\mathrm{gw}^T(\pi)$, is the maximum grid-width of a vertex of T. Finally, the *grid-width* of a permutation π, denoted by $\mathrm{gw}(\pi)$, is the minimum of $\mathrm{gw}^T(\pi)$ over all grid trees T of π.

It can be shown, by using the ideas of Ahal and Rabinovich [2], that the grid-width of a permutation π corresponds, up to a multiplicative constant, to the tree-width of the so-called adjacency graph G_π associated with π. This also implies that grid-width admits an efficient constant-factor approximation.

4.1 GT-recognizable classes

We shall now define a type of class whose recognition problem is tractable on inputs of bounded grid-width. Informally speaking, a class is GT-recognizable if its members can be recognized by a dynamic programming algorithm over their grid tree.

First, let us define an efficient way to encode merging of two interval families. A *merge description of interval families $\mathcal{I}_1$ and $\mathcal{I}_2$ into an interval family $\mathcal{I}$* is a pair (f, g), where

- $f : [|\mathcal{I}_1| + |\mathcal{I}_2|] \to \{1, 2\}$ encodes the interleaving of the intervals of $\mathcal{I}_1$ and $\mathcal{I}_2$, and
- g is a monotone function $[|\mathcal{J}|] \to [|\mathcal{I}_1| + |\mathcal{I}_2|]$ that describes first interval of each consecutive sequence of intervals that merges to a single interval.

Observe that the knowledge of the merge description together with the interval families $\mathcal{I}_1$ and $\mathcal{I}_2$ uniquely determines the resulting interval family $\mathcal{I}$.

For the following definitions, fix a permutation π of length n with a grid tree T. For any vertex v, let $\mathcal{I}_v$ be the unique minimal interval decomposition of $x(S_v^T)$ and similarly let $\mathcal{J}_v$ be the unique minimal interval decomposition of $y(S_v^T)$. Let u be a vertex of T with children v and w. A *merge description of vertex* u is then a pair (M_1, M_2), where

- M_1 is a merge description of the interval families $\mathcal{I}_v$ and $\mathcal{I}_w$ into $\mathcal{I}_u$, and
- M_2 is a merge description of the interval families $\mathcal{J}_v$ and $\mathcal{J}_w$ into $\mathcal{J}_u$.

It is easy to see that the shape of T (omitting the labels on its leaves) together with merge descriptions of its inner vertices uniquely determines both the original T and π. For technical reasons, we now allow grid trees to have unlabeled (empty) leaves, which represent an empty subpermutation. For an empty leaf v, both interval families $\mathcal{I}_v$ and $\mathcal{J}_v$ are just empty sets. We say that T is a *merge-labeled tree* if every inner vertex is labeled with its merge description, every non-empty leaf has label ϵ_0 and every empty leaf has label ϵ_1.

We say that a permutation class $\mathcal{C}$ is *grid tree* recognizable, or GT-recognizable for short, if there is an algorithm A that receives the grid-width g and outputs a tree automaton that recognizes $\mathcal{C}$ over merge-labeled trees of grid-width at most g.

A *tree automaton* over merge-labeled trees is a tuple $\mathcal{A} = (Q, \Delta, F)$, where Q is a set of states, $F \subseteq Q$ is a set of final states, and Δ is a set of transition rules of the form $(M, q_1, q_2) \to q$, for merge description M and states $q_1, q_2 \in Q$, and of rules of the form $\epsilon_i \to q$ for $i \in \{0, 1\}$ and $q \in Q$. A *run* of $\mathcal{A}$ on a merge-labeled tree T is simply T labeled with states from Q such that all the states together with their transitions are consistent with the rules of Δ. A run is *accepting* if its state q in the root of T belongs to the set of final states F.

As with NLOL and 2D-NLOL, the GT-recognizable classes are closed with respect to many important operations.

▶ **Lemma 4.** *If $\mathcal{C}$ and $\mathcal{D}$ are GT-recognizable classes, then the following classes are GT-recognizable as well:*

(a) *The classes $\mathcal{C} \cap \mathcal{D}$ and $\mathcal{C} \cup \mathcal{D}$.*
(b) *The classes $\mathcal{C}^r$, $\mathcal{C}^c$ and $\mathcal{C}^{-1}$.*
(c) *The classes $\mathcal{C}^{\oplus}$ and $\mathcal{C}^{\ominus}$, i.e., the sum-closure and skew-closure of $\mathcal{C}$.*
(d) *The classes $\mathcal{C} \oplus \mathcal{D}$ and $\mathcal{C} \ominus \mathcal{D}$.*
(e) *The class $\mathcal{C} \odot \mathcal{D}$.*

Moreover, it can be shown that any class determined by a finite set of minimal forbidden patterns is GT-recognizable.

Fix an input permutation π. Let A be a 2D-NLOL-recognizer and M its set of memory states. We call a point set E a *grid set* if it can be expressed as $E_x \times E_y$ for some $E_x, E_y \subseteq [n]$. A tuple (E, g) is a *grid set of positions* if E is a grid point set and $g : E \to M$. We say that (E, g) is *consistent* if for any two points $p = (p_x, p_y), r = (r_x, r_y) \in E$ such that $p_x \le r_x$ and $p_y \le r_y$, A can reach position $(p, g(p))$ from position $(r, g(r))$. The first lemma claims that if we have a box with prescribed states in the lower left and upper right corner, which constitute a reachable pair, then we can extend it to consistent grid set for arbitrary subgridding of the box. The second simply states that for a consistent grid set, we can exchange contents of any box as long as we do not violate reachability locally.

▶ **Lemma 5.** *Let $E = E_x \times E_y$ be a grid set, $e_1 \le e_2$ the minimal and maximal element of E_x and $f_1 \le f_2$ the minimal and maximal element of E_y. Let $m^{\llcorner}, m^{\urcorner} \in A$ be a pair of states such that (e_1, f_1) and (e_2, f_2) are reachable through $m^{\llcorner}$ and $m^{\urcorner}$. Then there is a function $g : E \to M$ such that (E, g) is consistent and moreover $g(e_1, f_1) = m^{\llcorner}$ and $g(e_2, f_2) = m^{\urcorner}$.*

▶ **Lemma 6.** *Let (E, g) be a grid set consistent over some subpermutation π' of permutation π, and $p = (p_1, p_2)$ and $r = (r_1, r_2)$ two its points such that $E \cap [p_1, p_2] \times [r_1, r_2]$ contains only the four points $(p_1, p_2), (r_1, p_2), (p_1, r_2), (r_1, r_2)$. Then replacing the subpermutation $\pi'|_{[p_1,r_1)\times[p_2,r_2)}$ with a different subpermutation σ of π does not violate the consistency property as long as the reachability is preserved for all the pairs among the points $(p_1, p_2), (r_1, p_2), (p_1, r_2), (r_1, r_2)$.*

We may now state and prove our main result.

▶ **Theorem 7.** *If $\mathcal{C}$ is a 2D-NLOL-recognizable class and $\mathcal{D}$ is a GT-recognizable class such that every $\pi \in \mathcal{D}$ has grid-width bounded by g, then $\mathcal{C} \odot \mathcal{D}$ is polynomially recognizable.*

Proof. Let the input be a permutation π of length n, let A be the 2D-NLOL-recognizer of $\mathcal{C}$ and $\mathcal{B}$ be the tree automaton recognizing $\mathcal{D}$ over merge-labeled trees of grid-with at most g. The general outline of our approach is fairly simple, we want to efficiently emulate $\mathcal{B}$ on all the subpermutations of π with grid-width at most g while at the same time simulating A on the remaining elements. Throughout this proof we shall use the color red to color the part belonging to $\mathcal{C}$ and blue for the part belonging to $\mathcal{D}$. Let M denote the set of possible memory states of A during computation on permutation of length n, and let N denote the set of states of $\mathcal{B}$. Observe that the size of M is at most n^c for some constant c and the size of N is at most $f(g)$ for a computable function f.

We shall define a polynomially bounded number of problems that can be effectively solved by recursion. A *problem* is a tuple $(\mathcal{I}, \mathcal{J}, \mathcal{Q}, s)$, where

- $\mathcal{I}$ and $\mathcal{J}$ are interval families of integers in $[n]$ each of size at most g,
- $\mathcal{Q} : \mathcal{I} \times \mathcal{J} \to M^4$ assigns four memory states of A to each pair of the intervals, and
- $s \in N$ is a possible state of the automaton $\mathcal{B}$.

There are at most n^{4g} choices for the intervals, at most n^{4cg^2} choices for the memory states and finally at most $f(g)$ choices for the states of the tree automaton $\mathcal{B}$, which makes the total number of problems at most $f(g)n^{4g+4g^2c}$.

We then say that a problem $(\mathcal{I}, \mathcal{J}, \mathcal{Q}, s)$ is *feasible* if there is a red-blue coloring of the subset of π that lies in the union of $\mathcal{I} \times [n]$ and $[n] \times \mathcal{J}$ with the following properties:

- the permutation π_B corresponding to the blue elements is contained in $\mathcal{I} \times \mathcal{J}$ and moreover, for every $I = [i_1, i_2] \in \mathcal{I}$ it holds that both (i_1, π_{i_1}) and (i_2, π_{i_2}) are colored blue, and similarly for every $J = [j_1, j_2] \in \mathcal{J}$ we have that both $(\pi^{-1}_{j_1}, j_1)$ and $(\pi^{-1}_{j_2}, j_2)$ are colored blue,
- π_B belongs to $\mathcal{D}$ and there exists its grid tree T of grid-width at most g whose root has its minimal interval decompositions identical to $\mathcal{I}$ and $\mathcal{J}$, and moreover, there is a run of the automaton $\mathcal{B}$ over the tree T that assigns the state s to the root of T, and
- for any two intervals $I = [i_1, i_2] \in \mathcal{I}$ and $J = [j_1, j_2] \in \mathcal{J}$ such that $\mathcal{Q}(I, J) = (m^{\llcorner}, m^{\lrcorner}, m^{\ulcorner}, m^{\urcorner})$, the grid set (E, l) that contains the points $(i_1, j_1), (i_2 + 1, j_1), (i_1, j_2 + 1), (i_2 + 1, j_2 + 1)$ with their respective states $m^{\llcorner}, m^{\lrcorner}, m^{\ulcorner}, m^{\urcorner}$, is consistent over the elements of π_R.

Let $m_0 \in M$ be the initial memory state of A, $m_F \in M$ be a memory state corresponding to a permutation in $\mathcal{C}$ and $s \in N$ be a final state of $\mathcal{B}$. We say that a problem $(\mathcal{I}$, $\mathcal{J}$, $\mathcal{Q}$, $s)$ is *initial* if $\mathcal{I}$ and $\mathcal{J}$ contain only single intervals $I = [i_1, i_2]$ and $J = [j_1, j_2]$ and $\mathcal{Q}(I, J) = (m^{\llcorner}, m^{\lrcorner}, m^{\ulcorner}, m^{\urcorner})$ such that the positions $((i_1, j_1), m^{\llcorner})$, $((i_2, j_1), m^{\lrcorner})$ and $((i_1, j_2), m^{\ulcorner})$ are reachable from $((0, 0), m_0)$, and $((n + 1, n + 1), m_F)$ is reachable from $((i_2 + 1, j_2 + 1), m^{\urcorner})$. It follows from the definition that π belongs to $\mathcal{C} \odot \mathcal{D}$ if and only if one of

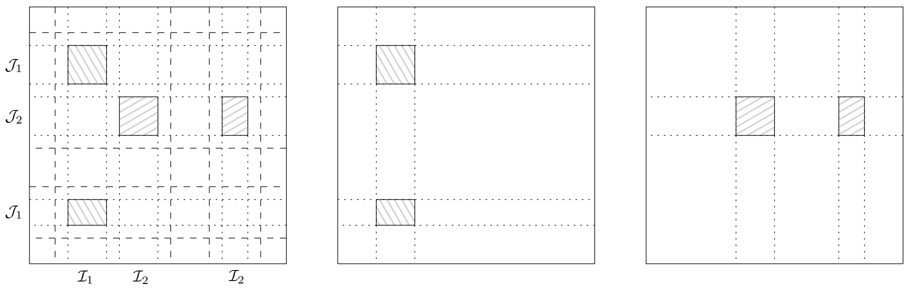


Figure 2 Decomposing problems into subproblems in Feasible. The original problem (left) and its possible subproblems (center and right).

the initial problems is feasible. Thus, we can decide membership if we compute the feasibility of all the initial problems since the additional conditions above are easily checkable.

We describe a recursive algorithm Feasible($\mathcal{I}$, $\mathcal{J}$, $\mathcal{Q}$, s) that takes a problem and either reports unsuccess or outputs some red-blue coloring of π restricted to $\mathcal{I} \times [n] \cup [n] \times \mathcal{J}$ that witnesses its feasibility. If we have a problem where both $\mathcal{I}$ and $\mathcal{J}$ contain only one interval, and the interval is in fact just a single point, then the feasibility of such problem is easily decidable. Otherwise, we recursively call Feasible on a pair of subproblems ($\mathcal{I}_1$, $\mathcal{J}_1$, $\mathcal{Q}_1$, s_1) and ($\mathcal{I}_2$, $\mathcal{J}_2$, $\mathcal{Q}_2$, s_2) with the following properties:

- $\mathcal{I}_1, \mathcal{I}_2$ are two disjoint non-empty interval families whose union is contained in $\mathcal{I}$, and $\mathcal{J}_1, \mathcal{J}_2$ two disjoint interval families whose union is contained in $\mathcal{J}$,
- $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are consistent with $\mathcal{Q}$, and
- $s_1, s_2 \in N$ are arbitrary.

See Figure 2. There are at most n^{4g} choices for the interval families. In order to bound the number of states, observe that we have $8g^2$ positions for the memory states and $f(g)$ states of $\mathcal{B}$, which gives us at most $f(g)^2 n^{8g^2}$ choices. This way, we defined at most $f(g)^2 n^{4g+8cg^2}$ pairs of strictly smaller subproblems and we call Feasible recursively on each of them.

We continue by describing the composition of outputs returned by Feasible on the subproblems $(\mathcal{I}_1, \mathcal{J}_1, \mathcal{Q}_1, s_1)$ and $(\mathcal{I}_2, \mathcal{J}_2, \mathcal{Q}_2, s_2)$ into a coloring returned by Feasible on the problem ($\mathcal{I}$, $\mathcal{J}$, $\mathcal{Q}$, s). If at least one of the recursive calls ends unsuccessfully we move to the next pair. Suppose that both of them are feasible and we have red-blue colorings of π restricted to the union of $\mathcal{I}_\alpha \times [n]$ and $[n] \times \mathcal{J}_\alpha$ for $\alpha \in \{0, 1\}$. Since we are trying to emulate the interval merging of a grid tree, we color all the remaining elements in the union of $\mathcal{I} \times [n]$ and $[n] \times \mathcal{J}$ red. Now we trivially check if the first condition of feasibility holds. In order to satisfy the second condition, it is sufficient to verify that $\mathcal{B}$ contains a transition $((M_1, M_2), s_1, s_2) \to s$ where M_1 and M_2 are the merge descriptions of the interval families $\mathcal{I}_1, \mathcal{I}_2$ into $\mathcal{I}$ and $\mathcal{J}_1, \mathcal{J}_2$ into $\mathcal{J}$. Note that in our case the union $\mathcal{I}_1$ and $\mathcal{I}_2$ might not be equal to $\mathcal{I}$ but we simply define the merge descriptions while forgetting the missing elements (and similarly for $\mathcal{J}$). This check takes at most $h(g)$ time for some computable function h depending on $\mathcal{D}$.

Finally, we need to check the third condition of feasibility. Fix some intervals $I \in \mathcal{I}$ and $J \in \mathcal{J}$ with $\mathcal{Q} = (m^{\llcorner}, m^{\lrcorner}, m^{\ulcorner}, m^{\urcorner})$. Since we have a coloring of $\mathcal{I} \times [n]$ and $[n] \times \mathcal{J}$, it suffices to check whether the corresponding grid set is consistent over the elements of π_R precisely as described in the condition. Simulating the nondeterministic recognizer on fixed input can be done in at most $O(n^{c+1})$ time, thus making the total time spent checking the third condition at most $O(g^2 n^{c+1})$. If all three verifications succeed we output the created coloring.

It follows that the total time spent computing FEASIBLE($\mathcal{I}$, $\mathcal{J}$, $\mathcal{Q}$, s), omitting the recursive calls, is $O(f(g)^3 h(g) g^2 n^{4g+8cg^2+c+c_2})$ where c_2 is a constant independent of g that captures the time spent per subproblem on enumerating all the possible subproblem pairs and testing the first condition. Therefore, the total time required to solve all the problems is at most $O(f'(g) n^{12cg^2+8g+c+c_2})$ where f' is some computable function.

Whenever FEASIBLE outputs a coloring of an initial problem, we verify all the conditions of feasibility and thus we obtain a coloring that witnesses $\pi \in \mathcal{C} \odot \mathcal{D}$. For the converse, suppose that $\pi \in \mathcal{C} \odot \mathcal{D}$ and we aim to show that we obtain a positive answer on the membership problem. Fix a red-blue coloring witnessing $\pi \in \mathcal{C} \odot \mathcal{D}$. Due to Lemma 5, there is a grid set (E, l) with $E = [n+1] \times [n+1]$ that is consistent with the accepting computation of A over π_R. We say that a problem ($\mathcal{I}$, $\mathcal{J}$, $\mathcal{Q}$, s) is *globally feasible* if the problem is feasible, there is an extension of some feasible coloring to the fixed coloring of the whole π and $\mathcal{Q}$ assigns precisely the memory states prescribed by (E, l). As we mentioned before, if π can be properly colored then there has to be some initial state that is globally feasible. We aim to show that for a globally feasible problem ($\mathcal{I}$, $\mathcal{J}$, $\mathcal{Q}$, s), FEASIBLE successfully outputs a feasible coloring which would therefore imply the correctness of the algorithm.

We prove this by induction on the size of $\mathcal{I}$ and $\mathcal{J}$. For any globally feasible problem with $\mathcal{I}$ and $\mathcal{J}$ such that both $\mathcal{I} \times \mathcal{J}$ contain a single point, FEASIBLE clearly outputs some coloring. For a larger globally feasible problem ($\mathcal{I}$, $\mathcal{J}$, $\mathcal{Q}$, s), we first describe how to split the problem into two specific subproblems that are also globally feasible. The splitting of the interval families is uniquely determined by the fixed coloring of the whole permutation together with the first condition of feasibility. Note that the first condition also ensures that we do not reach leaves of the grid tree corresponding to π_B before the interval families get trivial. Second condition determines the states of the tree automaton and we define $\mathcal{Q}$ to be consistent with the grid set (E, l). It is easy to see that these subproblems are also globally feasible and thus the subsequent calls of FEASIBLE return two partial colorings.

Finally, it remains to argue that it does not matter which feasible colorings we obtain from the recursion. Suppose that the subsequent calls of FEASIBLE returned feasible colorings different from our fixed coloring. However, here we can use the global feasibility together with Lemma 6 to see that we can replace the subpermutations box by box and the consistency is preserved. Therefore, the coloring obtained by joining the feasible colorings of the subproblems satisfies all the conditions and is returned by FEASIBLE. ◀

5 Hard cases of merge-recognition

Let us mention, without going into details, that we can also provide examples of merges $\mathcal{C} \odot \mathcal{D}$ whose recognition problem is NP-hard, even when the classes $\mathcal{C}$ and $\mathcal{D}$ are themselves determined by a single forbidden pattern. Specifically, we can prove the following result.

▶ **Theorem 8.** *For any simple permutation α of order at least 4, the recognition problem for the class $Av(\alpha) \odot Av(\alpha)$ is NP-complete.*

6 Concluding remarks and open problems

The complexity of many cases of ($\mathcal{C} \odot \mathcal{D}$)-recognition remains open. One natural question is to consider the merge of GT-recognizable classes that have bounded grid-width but do not belong to NLOL. Classes of this type include many important examples, such as the class $\mathrm{Av}(2413, 3142)$ of separable permutations, or the class $\mathrm{Av}(213)$ and its symmetries.

▶ **Open problem 1.** *What is the complexity of* $(\mathcal{C} \odot \mathcal{D})$*-recognition when* $\mathcal{C}$ *and* $\mathcal{D}$ *are any two (possibly identical) classes from the set* $\{Av(2413, 3142), Av(213), Av(231), Av(132), Av(312)\}$*?*

It is also natural to consider 'unbalanced' merges, when one of the two classes is very simple, e.g., the class Av(21) of increasing permutations. Our results imply that $(\mathcal{C} \odot \mathrm{Av}(21))$-recognition is tractable when $\mathcal{C}$ is in NLOL or when $\mathcal{C}$ is a GT class of bounded grid-width, but we know nothing about the remaining cases.

▶ **Open problem 2.** *For which classes* $\mathcal{C}$ *is the* $(\mathcal{C} \odot Av(21))$*-recognition polynomial?*

References

1 D. Achlioptas, J. I. Brown, D. G. Corneil, and M. S. O. Molloy. The existence of uniquely $-G$ colourable graphs. *Discrete Math.*, 179(1-3):1–11, 1998. `doi:10.1016/S0012-365X(97)00022-8`.

2 S. Ahal and Y. Rabinovich. On complexity of the subpattern problem. *SIAM J. Discrete Math.*, 22(2):629–649, 2008. `doi:10.1137/S0895480104444776`.

3 M. Albert and V. Jelínek. Unsplittable classes of separable permutations. *Electron. J. Combin.*, 23(2):Paper 2.49, 20, 2016.

4 M. Albert, J. Pantone, and V. Vatter. On the growth of merges and staircases of permutation classes. arXiv:1608.06969, 2016.

5 M. H. Albert. On the length of the longest subsequence avoiding an arbitrary pattern in a random permutation. *Random Structures Algorithms*, 31(2):227–238, 2007. `doi:10.1002/rsa.20140`.

6 V. E. Alekseev, A. Farrugia, and V. V. Lozin. New results on generalized graph coloring. *Discrete Math. Theor. Comput. Sci.*, 6(2):215–221, 2004.

7 D. Bevan, R. Brignall, A. Elvey Price, and J. Pantone. Staircases, dominoes, and the growth rate of 1324-avoiders. *Electronic Notes in Discrete Mathematics*, 61:123–129, 2017. The European Conference on Combinatorics, Graph Theory and Applications (EUROCOMB'17). `doi:10.1016/j.endm.2017.06.029`.

8 M. Bóna. A new upper bound for 1324-avoiding permutations. *Combin. Probab. Comput.*, 23(5):717–724, 2014. `doi:10.1017/S0963548314000091`.

9 M. Bóna. A new record for 1324-avoiding permutations. *Eur. J. Math.*, 1(1):198–206, 2015. `doi:10.1007/s40879-014-0020-6`.

10 P. Borowiecki. Computational aspects of greedy partitioning of graphs. *J. Comb. Optim.*, 35(2):641–665, 2018. `doi:10.1007/s10878-017-0185-2`.

11 P. Bose, J. F. Buss, and A. Lubiw. Pattern matching for permutations. *Inform. Process. Lett.*, 65(5):277–283, 1998. `doi:10.1016/S0020-0190(97)00209-3`.

12 J. I. Brown. The complexity of generalized graph colorings. *Discrete Appl. Math.*, 69(3):257–270, 1996. `doi:10.1016/0166-218X(96)00096-0`.

13 A. Claesson, V. Jelínek, and E. Steingrímsson. Upper bounds for the Stanley–Wilf limit of 1324 and other layered patterns. *J. Comb. Theory A*, 119:1680–1691, 2012.

14 T. Ekim, P. Heggernes, and D. Meister. Polar permutation graphs are polynomial-time recognisable. *European J. Combin.*, 34(3):576–592, 2013. `doi:10.1016/j.ejc.2011.12.007`.

15 A. Farrugia. Vertex-partitioning into fixed additive induced-hereditary properties is NP-hard. *Electron. J. Combin.*, 11(1):Research Paper 46, 9, 2004.

16 S. Guillemot and D. Marx. Finding small patterns in permutations in linear time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 82–101. ACM, New York, 2014. `doi:10.1137/1.9781611973402.7`.

17 V. Jelínek and M. Opler. Splittability and 1-amalgamability of permutation classes. *Discrete Math. Theor. Comput. Sci.*, 19(2):Paper No. 4, 14, 2017.

18 V. Jelínek and P. Valtr. Splittings and Ramsey properties of permutation classes. *Adv. Appl. Math.*, 63:41–67, 2015. doi:10.1016/j.aam.2014.10.003.

19 A. E. Kézdy, H. S. Snevily, and C. Wang. Partitioning permutations into increasing and decreasing subsequences. *J. Combin. Theory Ser. A*, 73(2):353–359, 1996.

20 V. Rutenburg. Complexity of generalized graph coloring. In *Mathematical foundations of computer science, 1986 (Bratislava, 1986)*, volume 233 of *Lecture Notes in Comput. Sci.*, pages 573–581. Springer, Berlin, 1986. doi:10.1007/BFb0016284.

21 V. Vatter. An Erdős-Hajnal analogue for permutation classes. *Discrete Math. Theor. Comput. Sci.*, 18(2):Paper No. 4, 5, 2016.

Solving Partition Problems Almost Always Requires Pushing Many Vertices Around

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

Christian Komusiewicz[1]
Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, Germany
komusiewicz@informatik.uni-marburg.de

Manuel Sorge[2]
Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer Sheva, Israel
sorge@post.bgu.ac.il

Erik Jan van Leeuwen
Department of Information and Computing Sciences, Utrecht University, The Netherlands
e.j.vanleeuwen@uu.nl

Abstract

A fundamental graph problem is to recognize whether the vertex set of a graph G can be bipartitioned into sets A and B such that $G[A]$ and $G[B]$ satisfy properties Π_A and Π_B, respectively. This so-called (Π_A, Π_B)-Recognition problem generalizes amongst others the recognition of 3-colorable, bipartite, split, and monopolar graphs. A powerful algorithmic technique that can be used to obtain fixed-parameter algorithms for many cases of (Π_A, Π_B)-Recognition, as well as several other problems, is the *pushing process*. For bipartition problems, the process starts with an "almost correct" bipartition (A', B'), and pushes appropriate vertices from A' to B' and vice versa to eventually arrive at a correct bipartition.

In this paper, we study whether (Π_A, Π_B)-Recognition problems for which the pushing process yields fixed-parameter algorithms also admit polynomial problem kernels. In our study, we focus on the first level above triviality, where Π_A is the set of P_3-free graphs (disjoint unions of cliques, or cluster graphs), the parameter is the number of clusters in the cluster graph $G[A]$, and Π_B is characterized by a set $\mathcal{H}$ of connected forbidden induced subgraphs. We prove that, under the assumption that $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$, (Π_A, Π_B)-Recognition admits a polynomial kernel if and only if $\mathcal{H}$ contains a graph of order at most 2. In both the kernelization and the lower bound results, we make crucial use of the pushing process.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms, Theory of computation → Algorithm design techniques

Keywords and phrases Fixed-parameter algorithms, Kernelization, Vertex-partition problems, Reduction rules, Cross-composition

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.51

[1] CK gratefully acknowledges support by the DFG, project MAGZ, KO 3669/4-1.
[2] MS gratefully acknowledges support by the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement number 631163.11, by the Israel Science Foundation (grant number 551145/14), and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement number 714704.

© Iyad Kanj, Christian Komusiewicz, Manuel Sorge, and Erik Jan van Leeuwen;
licensed under Creative Commons License CC-BY

1 Introduction

A graph G is a (Π_A, Π_B)-*graph*, for two hereditary graph properties Π_A, Π_B, if $V(G)$ can be partitioned into two sets A, B such that $G[A] \in \Pi_A$ and $G[B] \in \Pi_B$. We call (A, B) a (Π_A, Π_B)-*partition* of G. The (Π_A, Π_B)-Recognition problem is to recognize whether a given graph is a (Π_A, Π_B)-graph. This captures a wealth of famous problems, including the recognition of 3-colorable, bipartite, co-bipartite, and split graphs, and Π-Vertex Deletion, which asks for a partition (A, B) such that $G[A] \in \Pi$ and $G[B]$ has order at most k for some given k. In the most interesting (and NP-hard) cases [2, 13, 22], Π_A and Π_B are both characterized by a (not necessarily finite) set of forbidden connected induced subgraphs. In other words, Π_A and Π_B are each closed under the disjoint union of graphs in these cases.

Many such (Π_A, Π_B)-Recognition problems were shown fixed-parameter tractable by Kanj *et al.* [20], for example when Π_A is the class of graphs that is a disjoint union of k cliques, using parameter k. The central algorithmic idea that was employed in [20] is the *pushing process*. The algorithm empties the input graph, and adds vertices back one by one while maintaining a valid partition. Since adding a vertex might invalidate a previously valid partition, vertices are *pushed* from one part of the partition to the other part in the hope of obtaining a valid partition again. A similar algorithmic idea, known as iterative localization, was used earlier by Heggernes *et al.* [19] to show the fixed-parameter tractability of computing the cochromatic number of perfect graphs and the stabbing number of disjoint rectangles with axes-parallel lines (using the standard parameters). Iterative localization was also applied in follow-up work related to the cochromatic number [21].

A crucial ingredient in applying the pushing process is to understand the *avalanches* caused by this process. For (Π_A, Π_B)-Recognition, an avalanche is triggered when a vertex is pushed to A; this may imply that several other vertices must be pushed to B, which, in turn, triggers the pushing of yet more vertices to A, and so on. Similar effects are visible in the aforementioned cochromatic number and rectangle stabbing number problems. The contribution of the previous works [19, 20, 21] was to bound the depth of this process by some function of the parameter, leading to fixed-parameter algorithms. However, such a bound does not provide an answer to the question of which vertices trigger avalanches and their continued rolling, and whether the number of such vertices can somehow be limited.

This question can be naturally formalized in terms of the kernelization complexity of problems to which the pushing process applies. A kernel reduces the size of the graph and thus directly reduces the number of vertices triggering or being affected by avalanches when an algorithm based on the pushing process is applied to the kernelized instance. In previous work, Kolay *et al.* [21] studied the kernelization complexity of computing the cochromatic number of a perfect graph G, which is the smallest number $k = r + \ell$ such that $V(G)$ can be partitioned into r sets that each induces a clique and ℓ sets that each induces an edgeless graph. This problem has a parameterized algorithm using iterative localization (*i.e.*, a pushing process) [19], but Kolay *et al.* [21] showed that, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$, this problem does not admit a polynomial kernel parameterized by $r + \ell$. This suggests that, for this problem, one cannot control the number of vertices affected by avalanches. The kernelization complexity of (Π_A, Π_B)-Recognition, however, has not been studied so far. Hence, it is open whether avalanches can be controlled to affect few vertices in this case.

Our Result. We study the kernelization complexity of (Π_A, Π_B)-Recognition through the lens of the pushing process. To this end, we consider the first level above triviality of the problem. When Π_A is characterized by a forbidden induced subgraph of order 2, then

(Π_A, Π_B)-Recognition can be solved in linear time [16], and thus we focus on the NP-hard case when the forbidden induced subgraph has order 3 [2, 13, 22]. In particular, we let Π_A be the class of so-called *cluster graphs*. These are the graphs that contain no P_3 – the (simple) path on three vertices – as an induced subgraph, or equivalently, graphs that are disjoint unions of complete graphs. This leads to the following problem:

Cluster-Π-Partition
Input: A graph $G = (V, E)$.
Question: Is there a partition (A, B) of V such that $G[A]$ is a cluster graph and $G[B] \in \Pi$?

Cluster-Π-Partition generalizes the recognition problem of many graph classes, such as the recognition of *monopolar graphs* [6, 9, 8, 23] (Π is the set of edgeless graphs), 2-subcolorable graphs [5, 15, 18, 24] (Π is the set of cluster graphs), and several others [1, 4, 7]. Unfortunately, Cluster-Π-Partition is NP-hard in these special cases, and in general when Π is characterized by a set of connected forbidden induced subgraphs [2, 13, 22]. Hence, we consider the number k of clusters in the cluster graph $G[A]$ as a parameter, and study the pushing process with respect to this parameter.

Our result gives a complete characterization of the kernelization complexity of Cluster-Π-Partition through a deeper understanding of the pushing process. We show that, while for a specific Π the pushing process can be used to witness a small vertex set of size $k^{O(1)}$ containing the vertices affected by avalanches, for all other Π, such a set of polynomial size is unlikely to exist. Formally, we show that:

▶ **Theorem 1.1.** *Let Π be a graph property characterized by a (not necessarily finite) set $\mathcal{H}$ of connected forbidden induced subgraphs. Then unless* NP ⊆ coNP/poly, *Cluster-Π-Partition parameterized by the number k of clusters in the cluster graph $G[A]$ admits a polynomial kernel if and only if $\mathcal{H}$ contains a graph of order at most 2.*

The positive result corresponds to the recognition of monopolar graphs. Indeed, the graph properties with forbidden induced subgraphs of order 2 are "being edgeless" and "being nonedge-less", but the latter is not characterized by connected forbidden induced subgraphs.

The pushing process and a deeper understanding of the avalanches it causes are indeed central to both directions of the above result. In the proof of the positive result, we first perform a set of data reduction rules to identify some vertices that are part of A or B in *any* partition (A, B) of $V(G)$ such that $G[A]$ is a cluster graph with at most k clusters and $G[B]$ is edgeless. More importantly, these rules restrict the combinatorial properties of the graph induced by the remaining vertices. With these restrictions, it becomes possible to model the avalanches that occur using a bipartite graph. This graph enables two further reduction rules that lead to the polynomial kernel.

For the negative result, we observe that the bipartite graph constructed in the kernel is closely tied to the deterministic behavior of the pushing process for monopolar graphs: when an edge in $G[B]$ is created by pushing a vertex to B, the other endpoint of the edge must be pushed to A (recall that $G[B]$ must become edgeless). This limits the avalanches. However, for more complex properties Π_B, such a simple correspondence no longer exists. In particular, when the forbidden induced subgraphs have order at least 3, pushing a vertex to B may create a forbidden induced subgraph in $G[B]$ that can be repaired in at least two different ways. Then the pushing process starts to behave nondeterministically, and the avalanches grow beyond control. We exploit this intuition to exclude the existence of a polynomial kernel, unless NP ⊆ coNP/poly, by providing a cross-composition.

Other Parameterizations. One might consider two other parameters: the size of a largest cluster in $G[A]$ and the size of one of the sides. The size of a largest cluster in $G[A]$ will not lead to tractability, as Cluster-Π-Partition is NP-hard on subcubic graphs, even when Π is the set of edgeless graphs [23]. Thus, we consider the number k of vertices in the graph $G[B]$, even for the broader (Π_A, Π_B)-Recognition problem, observing a general result:

▶ **Theorem 1.2.** (♠)[3] (Π_A, Π_B)-Recognition *has a kernel of size $\mathcal{O}(k^d)$ parameterized by k, the maximum size of B, when Π_A can be characterized by a collection $\mathcal{H}$ of forbidden induced subgraphs, each of size at most d, and Π_B is hereditary.*

We obtain the following better bound in terms of the number of vertices for Cluster-Π_Δ-Partition, the restriction of Cluster-Π-Partition to the case when all graphs containing a vertex of degree at least $\Delta + 1$ are forbidden induced subgraphs of Π.

▶ **Theorem 1.3.** (♠) Cluster-Π_Δ-Partition *parameterized by k, the maximum size of B, has an $\mathcal{O}((\Delta^2 + 1) \cdot k^2)$-vertex kernel.*

Preliminaries. We follow standard graph-theoretic notation [11]. For $\ell \in \mathbb{N}$, we use $[\ell]$ to denote $\{1, 2, \ldots, \ell\}$. Let $v \in V(G)$ and $X, Y \subseteq V(G)$. We say v is *adjacent to* X if v is adjacent to at least one vertex in X. We say X *is adjacent to* Y if there exists $x \in X$ that is adjacent to Y. We say a partition (A, B) of $V(G)$ is a *cluster-*Π *partition* if (1) $G[A]$ is a cluster graph and (2) $G[B] \in \Pi$. A *monopolar partition* of a graph G is a partition of $V(G)$ into a cluster graph and an independent set. Monopolar Recognition asks, given a graph G and an integer k, whether G admits a monopolar partition (A, B) such that the number of clusters in the cluster graph $G[A]$ is at most k. For an instance (G, k) of Monopolar Recognition, a monopolar partition of G is *valid* if the number of clusters in the cluster graph of the partition is at most k. For relevant definitions of parameterized complexity, e.g. polynomial problem kernels, see [12, 10]. Let Q be a language and (P, κ) a parameterized problem, *i.e.*, P is a language and $\kappa \colon \Sigma^* \to \mathbb{N}$ a parameterization. An *or-cross-composition* from Q into (P, κ) is a polynomial-time algorithm that, given t instances $q_1, \ldots, q_t \in \Sigma^*$ of Q, computes an instance $r \in \Sigma^*$ such that $\kappa(r) \le \operatorname{poly}(\log t + \max_{i=1}^{t} |q_i|)$, and $r \in P$ if and only if $q_i \in Q$ for some $i \in [t]$. If there is an or-cross-composition from an NP-hard language into (P, κ), then there is no polynomial-size problem kernel for (P, κ) unless NP ⊆ coNP/poly [17, 3].

2 A Polynomial Kernel for Monopolar Recognition Parameterized by the Number of Clusters

The outline of the kernelization algorithm is as follows. First, we compute a decomposition of the input graph into sets of vertex-disjoint maximal cliques which we call a *clique decomposition*. This decomposition is used and updated throughout the data-reduction procedure. We also maintain sets of vertices that are determined to belong to A or B. We first apply a sequence of reduction rules whose aim is roughly to bound the number of cliques and the number of edges between the cliques in the decomposition, and to restrict the structure of edges between cliques. Then, we build an auxiliary graph to model how the placement of a vertex in A or B implies an avalanche of placements of vertices in A and B. If this avalanche creates too many clusters in A, then this determines the placement of certain

[3] Due to lack of space, proofs of statements marked with (♠) are omitted.

vertices in A or B, and triggers another reduction rule. If this reduction rule does not apply anymore, then the size of the auxiliary graph is bounded, which in turn, helps bounding the size of the instance.

Clique Decompositions. Say that a clique C is a *large clique* if $|C| \geq 3$, an *edge clique* if $|C| = 2$ (*i.e.*, C is an edge), and a *vertex clique* if $|C| = 1$ (*i.e.*, C consists of a single vertex). Let (G, k) be an instance of Monopolar Recognition. Suppose that $A_{\mathsf{true}} \subseteq V(G)$ and $B_{\mathsf{true}} \subseteq V(G)$ are subsets of vertices that have been determined to be in A and B, respectively, in any valid monopolar partition of (G, k). We define a decomposition $(C_1, \ldots, C_r)$ of $V(G) \setminus (A_{\mathsf{true}} \cup B_{\mathsf{true}})$, referred to as a *nice clique decomposition*, that partitions this set into vertex-disjoint cliques $C_1, \ldots, C_r$, $r \geq 1$, such that the tuple $(C_1, \ldots, C_r)$ satisfies the following properties:

(i) In the decomposition tuple $(C_1, \ldots, C_r)$, the large cliques appear before the edge cliques, and the edge cliques, in turn, appear before the vertex cliques; that is, for each large clique C_i and for each edge or vertex clique C_j we have $i < j$, and for each edge clique C_i and for each vertex clique C_j we have $i < j$.

(ii) Each clique C_i, $i \in [r-1]$, is maximal in $\bigcup_{j=i}^{r} C_j$; that is, there does not exist a vertex $v \in \bigcup_{j=i+1}^{r} C_j$ such that $C_i \cup \{v\}$ is a clique.

(iii) The subgraph of G induced by the union of the edge cliques and vertex cliques does not contain any large clique.

The following fact is implied by property (ii) above:

▶ **Fact 2.1.** *The vertex cliques in a nice clique decomposition form an independent set in G.*

▶ **Lemma 2.2.** (♠) *A nice clique decomposition of G can be computed in $\mathcal{O}(nm)$ time.*

Let (G, k) be an instance of Monopolar Recognition. We initialize $A_{\mathsf{true}} = B_{\mathsf{true}} = \emptyset$, $V' = V(G) \setminus (A_{\mathsf{true}} \cup B_{\mathsf{true}})$, and we compute a nice clique decomposition $(C_1, \ldots, C_r)$ of V'. We will then apply reduction rules to simplify the instance (G, k). During this process, we may identify vertices in V' to be added to A_{true} or B_{true}. At any point in the process, we will maintain a partition $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \ldots, C_r)$ of $V(G)$ such that (1) $A_{\mathsf{true}} \subseteq A$ and $B_{\mathsf{true}} \subseteq B$ for any valid monopolar partition (A, B) of $V(G)$, and (2) $(C_1, \ldots, C_r)$ is a nice clique decomposition of $V' = V(G) \setminus (A_{\mathsf{true}} \cup B_{\mathsf{true}})$; we call such a partition $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \ldots, C_r)$ a *normalized partition* of $V(G)$.

Basic Reduction Rules. We now describe our basic set of reduction rules. After the application of a reduction rule, a normalized partition may change as the result of moving vertices from $\bigcup_{i=1}^{r} C_i$ to $A_{\mathsf{true}} \cup B_{\mathsf{true}}$, and we will need to compute a nice clique decomposition of the resulting (new) set $V(G) \setminus (A_{\mathsf{true}} \cup B_{\mathsf{true}})$. However, a vertex that has been moved to A_{true} (resp. B_{true}) will remain in A_{true} (resp. B_{true}). When a reduction rule is applied, we assume that no reduction rule preceding it is applicable. The following rule is straightforward:

▶ **Reduction Rule 2.3.** *Let $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \ldots, C_r)$ be a normalized partition of $V(G)$. If A_{true} is not a cluster graph with at most k clusters, or B_{true} is not an independent set, then reject the instance (G, k).*

The following rule is correct because, for every monopolar partition (A, B) of G, $B_{\mathsf{true}} \subseteq B$ and B is an independent set.

▶ **Reduction Rule 2.4.** *Let $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \ldots, C_r)$ be a normalized partition of $V(G)$. If there is a vertex $v \in V(G) \setminus (A_{\mathsf{true}} \cup B_{\mathsf{true}})$ that is adjacent to B_{true} then set $A_{\mathsf{true}} = A_{\mathsf{true}} \cup \{v\}$.*

The following rule is correct, since $A_{\mathsf{true}} \subseteq A$ for every monopolar partition (A, B) of G:

▶ **Reduction Rule 2.5.** *Let $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \ldots, C_r)$ be a normalized partition of $V(G)$. If there is a vertex $v \in V(G) \setminus (A_{\mathsf{true}} \cup B_{\mathsf{true}})$ that is either (1) adjacent to two clusters in A_{true}, or (2) adjacent to a cluster C in A_{true} but not to all the vertices in C, then set $B_{\mathsf{true}} = B_{\mathsf{true}} \cup \{v\}$.*

The proof of the following reduction rule is straightforward, after recalling that the vertex cliques induce an independent set in G (Fact 2.1), and observing that no two vertices of an independent set can belong to the same cluster in a cluster graph:

▶ **Reduction Rule 2.6.** *Let $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \ldots, C_r)$ be a normalized partition of $V(G)$. If there is a vertex $v \in V(G) \setminus (A_{\mathsf{true}} \cup B_{\mathsf{true}})$ with more than k neighbors that are vertex cliques, then set $A_{\mathsf{true}} = A_{\mathsf{true}} \cup \{v\}$.*

The next two reduction rules restrict the number and type of edges incident to large cliques.

▶ **Reduction Rule 2.7.** *Let $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \ldots, C_r)$ be a normalized partition of $V(G)$. If there exists a vertex $v \in V(G) \setminus (A_{\mathsf{true}} \cup B_{\mathsf{true}})$ and a large clique C_i such that $1 < |N(v) \cap C_i| \leq |C_i| - 1$, then set $A_{\mathsf{true}} = A_{\mathsf{true}} \cup (N(v) \cap C_i)$.*

Proof. Since $1 < |N(v) \cap C_i| \leq |C_i| - 1$, v has at least two neighbors $u, w \in C_i$ and at least one nonneighbor $x \in C_i$. If a vertex $z \in N(v) \cap C_i$ is in B, for any valid monopolar partition (A, B) of $V(G)$, then since B is an independent set, it follows that $C_i - \{z\} \subseteq A$. In particular, v is in A, at least one of u, w, say u, is in A, and x is in A. But this implies that (v, u, x) forms an induced P_3 in A, contradicting that A is a cluster graph. ◀

▶ **Reduction Rule 2.8.** (♠) *Let $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \ldots, C_r)$ be a normalized partition of $V(G)$, and let C_i, C_j, $i < j$, be two cliques such that C_i is a large clique and C_j is either a large clique or an edge clique. If there are at least two edges between C_i and C_j then one of the following reductions, considered in the listed order, is applicable:*

Case (1) *There are two edges uu' and vv', where $u, v \in C_i$ and $u', v' \in C_j$, such that $u \neq v$ and $u' \neq v'$. Let $w \in C_i$ be such that $w \notin \{u, v\}$ (note that w exists because $|C_i| \geq 3$). Set $A_{\mathsf{true}} = A_{\mathsf{true}} \cup \{w\}$.*

Case (2) *$N(C_j) \cap C_i = \{v\}$. Set $B_{\mathsf{true}} = B_{\mathsf{true}} \cup \{v\}$.*

We can now bound the number of large cliques and edge cliques in yes-instances.

▶ **Reduction Rule 2.9.** (♠) *Let (G, k) be an instance of* Monopolar Recognition, *and let $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \ldots, C_r)$ be a normalized partition of $V(G)$. If in $(C_1, \ldots, C_r)$ either the number of large cliques is more than k, or the number of large cliques plus the number of edge cliques is more than $2k$, then reject the instance (G, k).*

▶ **Reduction Rule 2.10.** (♠) *Let $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \ldots, C_r)$ be a normalized partition of $V(G)$, let C be a cluster in A_{true}, and let C_i, $i \in [r]$, be a large clique. If $v \in C_i$ is such that: (1) v is the only vertex in C_i that is adjacent to C, or (2) v is the only vertex in C_i that is not adjacent to C, then set $B_{\mathsf{true}} = B_{\mathsf{true}} \cup \{v\}$.*

▶ **Reduction Rule 2.11.** (♠) *Let (G, k) be an instance of* Monopolar Recognition, *and let $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \ldots, C_r)$ be a normalized partition of $V(G)$. If either (1) B_{true} contains more than $k + 1$ vertices or (2) there exists a cluster in A_{true} that is not a singleton, then reduce the instance (G, k) to an instance (G', k) with G' constructed as follows. Let $V(G') = V_1 \cup V_2 \cup V_3$, where $V_1 = \{u_C \mid C \text{ is a cluster in } A_{\mathsf{true}}\}$, $V_2 = \{v_1, \ldots, v_{k+1}\}$, and $V_3 = C_1 \cup \cdots \cup C_r$; and $E(G') = \{vu_C \mid v \in V_2 \wedge u_C \in V_1\} \cup \{vu_C \mid v \in V_3 \wedge u_C \in$*

$V_1 \wedge v$ is adjacent to $C\}$. That is, G' is constructed from G by introducing $k+1$ new vertices, replacing each cluster C in A_{true} (if any) by a single vertex u_C whose neighborhood is the neighborhood of C in $C_1, \ldots, C_r$ plus the $k+1$ new vertices, and keeping $C_1, \ldots, C_r$ the same.

If Reduction Rule 2.11 is applied, then after its application, we set A_{true} to V_1 and B_{true} to $\{v_1, \ldots, v_{k+1}\}$. Note that in any valid monopolar partition (A, B) of the graph resulting from the application of Reduction Rule 2.11, each vertex in V_1 must be in A, being adjacent to the $k+1$ independent set vertices $v_1, \ldots, v_{k+1}$, whereas the vertices $v_1, \ldots, v_{k+1}$ can be safely assumed to be in B since their only neighbors are in $V_1 \subseteq A$.

Modeling the Pushing Process by a Bipartite Graph. We now have bounded the number of large and edge cliques, and the size of A_{true} and B_{true}. It remains to bound the size of the large cliques and the number of vertex cliques. The challenge here is that we need to identify vertices such that putting them in A or B will eventually, after a series of pushes, lead either to the creation of too many clusters in A, or to the addition of two adjacent vertices in B. To model the avalanche of pushes to A or B, we introduce the following auxiliary graph.

▶ **Definition 2.12.** For a normalized partition $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \ldots, C_r)$ of $V(G)$, we define the auxiliary bipartite graph Λ as follows. The vertex set of Λ is $V(\Lambda) = V_C \cup V_I$, where V_C is the set of all vertices in the large cliques in $C_1, \ldots, C_r$, and V_I is the set of all vertices in the vertex cliques in $C_1, \ldots, C_r$. The edge set of Λ is $E(\Lambda) = \{uv \in E(G) \mid u \in V_C \text{ and } v \in V_I\}$; that is, $E(\Lambda)$ consists of precisely the edges in $E(G)$ that are between V_C and V_I.

Recall that V_I is an independent set in G by Fact 2.1. For a vertex $v \in V(\Lambda)$, we write $N_\Lambda(v)$ for the set of neighbors of v in Λ. We have the following lemma:

▶ **Lemma 2.13.** *Let $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \ldots, C_r)$ be a normalized partition of $V(G)$ and consider the graph $\Lambda = (V(\Lambda), E(\Lambda))$. Then the maximum degree of Λ, $\Delta(\Lambda)$, is at most k.*

Proof. For every vertex $v \in V_C$, we have $|N_\Lambda(v)| \leq k$ because Reduction Rule 2.6 is inapplicable. By property (ii) of a nice decomposition and the inapplicability of Reduction Rule 2.7, every vertex clique that is adjacent to a large clique C is adjacent to exactly one vertex in C. Since by Reduction Rule 2.9 the number of large cliques is at most k, every vertex in V_I, which is a vertex clique by definition of V_I, has at most k neighbors in V_C. Therefore, for every vertex $v \in V_I$, we have $|N_\Lambda(v)| \leq k$. ◀

For two vertices $u, v \in V(\Lambda)$, write $dist_\Lambda(u, v)$ for the length of a shortest path between u and v in Λ. For a vertex $v \in V(\Lambda)$ and $i \in \{0, \ldots, n\}$, define $N^i(v) = \{u \in V(\Lambda) \mid dist_\Lambda(u, v) = i\}$. Write $\bar{0}_n$ (resp. $\bar{1}_n$) for the set of even (resp. odd) integers in $\{0, \ldots, n\}$.

▶ **Lemma 2.14.** *Let $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \ldots, C_r)$ be a normalized partition of $V(G)$, let $\Lambda = (V(\Lambda), E(\Lambda))$ be the associated auxiliary graph, and let (A, B) be any valid monopolar partition of G.*

(i) *For each $v \in V_C$: If $v \in B$ then $N_\Lambda(v) \subseteq A$.*
(ii) *For each $v \in V_I$: If $v \in A$ then $N_\Lambda(v) \subseteq B$.*
(iii) *For each $v \in V_C$: If $v \in B$ then $N^i_\Lambda(v) \subseteq B$ for $i \in \bar{0}_n$, and $N^i_\Lambda(v) \subseteq A$ for $i \in \bar{1}_n$.*
(iv) *For each $v \in V_I$: If $v \in A$ then $N^i_\Lambda(v) \subseteq A$ for $i \in \bar{0}_n$, and $N^i_\Lambda(v) \subseteq B$ for $i \in \bar{1}_n$.*

Proof. (i): This trivially follows because B is an independent set.

(ii): Suppose that $v \in V_I$ is in A, and let $u \in N_\Lambda(v)$. Then $u \in V_C$ because Λ is bipartite, and hence, by definition, u belongs to a large clique C_i for some $i \in [r]$. Suppose, to get a contradiction, that $u \in A$. Since C_i is a large clique, and hence $|C_i| \geq 3$, there exists a vertex $w \neq u$ in C_i such that $w \in A$. By property (ii) of the nice decomposition $(C_1, \dots, C_r)$ and the inapplicability of Reduction Rule 2.7, $\{v, w\} \notin E(G)$. But this implies that (v, u, w) is an induced P_3 in A, contradicting that A is a cluster graph. It follows that $N_\Lambda(v) \subseteq B$.

(iii): This follows by repeated alternating applications of (i) and (ii) above.

(iv): This follows by repeated alternating applications of (ii) and (i) above. ◀

▶ **Reduction Rule 2.15.** (♠) *Let $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \dots, C_r)$ be a normalized partition of $V(G)$, and let $\Lambda = (V(\Lambda), E(\Lambda))$ be the associated auxiliary graph.*

(i) *For any vertex $v \in V_C$: If either $\bigcup_{i \in \bar{0}_n} N^i_\Lambda(v)$ contains two adjacent (in G) vertices or $|\bigcup_{i \in \bar{1}_n} N^i_\Lambda(v)| > k$, then set $A_{\mathsf{true}} = A_{\mathsf{true}} \cup \{v\}$.*

(ii) *For any vertex $v \in V_I$: If either $|\bigcup_{i \in \bar{0}_n} N^i_\Lambda(v)| > k$ or $\bigcup_{i \in \bar{1}_n} N^i_\Lambda(v)$ contains two adjacent (in G) vertices, then set $B_{\mathsf{true}} = B_{\mathsf{true}} \cup \{v\}$.*

Proof. (i) Let $v \in V_C$, and suppose that either $\bigcup_{i \in \bar{0}_n} N^i_\Lambda(v)$ contains two adjacent vertices or $|\bigcup_{i \in \bar{1}_n} N^i_\Lambda(v)| > k$. If $v \in B$ for any valid partition (A, B) of G, then by part (iii) of Lemma 2.14, it would follow that $\bigcup_{i \in \bar{0}_n} N^i_\Lambda(v) \subseteq B$ and $\bigcup_{i \in \bar{1}_n} N^i_\Lambda(v) \subseteq A$. In either case this contradicts that (A, B) is valid partition of G: If $\bigcup_{i \in \bar{0}_n} N^i_\Lambda(v)$ contains two adjacent vertices, then B is not an independent set, and if $|\bigcup_{i \in \bar{1}_n} N^i_\Lambda(v)| > k$ then A contains more than k clusters since $\bigcup_{i \in \bar{1}_n} N^i_\Lambda(v)$ induces an independent set in G.

(ii) The proof follows along the same lines as the proof of (i) (♠). ◀

▶ **Definition 2.16.** Let $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \dots, C_r)$ be a normalized partition of $V(G)$, and let $\Lambda = (V(\Lambda), E(\Lambda))$ be the associated auxiliary graph. From each large clique C_i, $i \in [r]$, fix three vertices u_i, v_i, w_i; define $V_{\mathsf{fixed}} = \{u_i, v_i, w_i \mid C_i \text{ is a large clique}\}$ to be the set of all fixed vertices. Define $V_{\mathsf{edge}} = \{u \mid u \text{ is contained in some edge clique } C_i\}$ to be the set of vertices of the edge cliques, define $N_{\mathsf{edge}} = N(V_{\mathsf{edge}}) \cap V(\Lambda)$ to be the neighbors of V_{edge} in $V(\Lambda)$, and define $N^\cup_{\mathsf{edge}} = \bigcup_{v \in N_{\mathsf{edge}}} \bigcup_{i \le n} N^i_\Lambda(v)$ to be the set of all vertices in $V(\Lambda)$ that are reachable in Λ from the vertices in N_{edge}. Define $V_{\mathsf{inter}} = \{u, v \mid u \in C_i \wedge v \in C_j \wedge i \neq j \wedge uv \in E(G) \wedge (C_i, C_j \text{ are large cliques})\}$ to be the set of endpoints of edges between large cliques, and define $N^\cup_{\mathsf{inter}} = \bigcup_{v \in V_{\mathsf{inter}}} \bigcup_{i \le n} N^i_\Lambda(v)$ to be the set of all vertices in $V(\Lambda)$ that are reachable in Λ from the vertices in V_{inter}. Finally, let $V_{\mathsf{rep}} = A_{\mathsf{true}} \cup B_{\mathsf{true}} \cup V_{\mathsf{fixed}} \cup N^\cup_{\mathsf{inter}} \cup V_{\mathsf{edge}} \cup N^\cup_{\mathsf{edge}}$.

▶ **Reduction Rule 2.17.** (♠) *Let (G, k) be an instance of* Monopolar Recognition, *and let $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \dots, C_r)$ be a normalized partition of $V(G)$. Let V_{rep} be as defined in Definition 2.16. Set $G = G[V_{\mathsf{rep}}]$.*

We now give the polynomial kernel whose existence was promised in Theorem 1.1.

▶ **Theorem 2.18.** Monopolar Recognition *has a kernel of size at most $9k^4 + 9k + 1$ which can be computed in $\mathcal{O}(n^2 m)$ time.*

Proof. Given an instance (G, k) of Monopolar Recognition, we apply Reduction Rules 2.3–2.17 exhaustively to (G, k). Clearly, the above rules can be applied in polynomial time. Let (G', k') be the resulting instance, let $(A_{\mathsf{true}}, B_{\mathsf{true}}, C_1, \dots, C_r)$ be a normalized partition of $V(G')$ with respect to which none of Reduction Rules 2.3–2.17 applies, and let $\Lambda = (V(\Lambda), E(\Lambda))$ be the auxiliary graph. Note that, by Reduction Rule 2.17, $V(G') = V_{\mathsf{rep}} = A_{\mathsf{true}} \cup B_{\mathsf{true}} \cup V_{\mathsf{fixed}} \cup N^\cup_{\mathsf{inter}} \cup V_{\mathsf{edge}} \cup N^\cup_{\mathsf{edge}}$. By Reduction Rule 2.9, the number of large cliques is at most k, and the number of edge cliques is at most $2k$. It follows that $|V_{\mathsf{fixed}}| \le 3k$ and $|V_{\mathsf{edge}}| \le 4k$. For a vertex $v \in V_{\mathsf{edge}}$, by Reduction Rule 2.6, v has at most k neighbors in V_I. Moreover, by Reduction Rule 2.8, v can have at most k neighbors in V_C, and therefore, $|N_\Lambda(v)| \le 2k$, and $|N_{\mathsf{edge}}| \le 4k \cdot 2k = 8k^2$. Since Reduction Rule 2.15 does not apply and $\Delta(\Lambda) \le k$ by Lemma 2.13, we have that, for any $v \in V(\Lambda)$, we have

$|\bigcup_{i\le n} N^i_\Lambda(v)| \le \Delta(\Lambda)\cdot k \le k^2$. This implies that $|N^\cup_{\mathsf{edge}}| \le 8k^2\cdot k^2 \le 8k^4$. Now since the number of large cliques is at most k, by Reduction Rule 2.8, it follows that $|V_{\mathsf{inter}}| \le \binom{k}{2} < k^2$. Since for a vertex $v \in V(\Lambda)$ we have $|\bigcup_{i\le n} N^i_\Lambda(v)| \le k^2$ as argued above, it follows that $|N^\cup_{\mathsf{inter}}| \le k^4$. Since $|A_{\mathsf{true}}| \le k$ and $|B_{\mathsf{true}}| \le k+1$, putting everything together, we conclude that the number of vertices in $V(G')$, $|V_{\mathsf{rep}}|$, is at most $k+k+1+3k+k^4+4k+8k^4 \le 9k^4+9k+1$. The running time proof is omitted (♠). ◀

3 Kernel-size lower bound

This section is dedicated to proving the "only if" direction of Theorem 1.1, which, together with Theorem 2.18, completes its proof. In particular, we prove the following:

▶ **Theorem 3.1.** *Let Π be a graph property characterized by a (not necessarily finite) set $\mathcal{H}$ of connected forbidden induced subgraphs, each of order at least* 3*. Then unless* NP $\subseteq$ coNP/poly*,* Cluster-Π-Partition *parameterized by the number k of clusters in the cluster graph $G[A]$ does not admit a polynomial kernel.*

Throughout, let Π be any graph property satisfying the conditions of Theorem 3.1. We show Theorem 3.1 by giving a cross-composition from the NP-hard problem Colorful Independent Set [14]. Herein, we are given a graph $G = (V, E)$, $k \in \mathbb{N}$, and a proper k-coloring $c\colon V \to \{1, \dots, k\}$; the question is whether there is an independent set with k vertices in G that contains exactly one vertex of each color. In the remainder of this section, we explain the construction behind the cross-composition and prove its correctness. We start by describing the intuition behind the construction, and why the avalanches in this case cannot be contained.

In contrast to Monopolar Recognition, the avalanches caused by the pushing process for the general Cluster-Π-Partition problem are much more uncontrollable: If some push to the Π-side B creates a forbidden induced subgraph M for Π in $G[B]$, we can repair the partition and "break" M by moving any vertex of M to the cluster graph side A. However, each move of a vertex in M may lead – through further necessary pushes from A to B – to distinct forbidden induced subgraphs in $G[B]$, again with multiple possible ways of breaking them in order to repair the partition. These avalanches cannot be contained, and lead to many possible paths along which they can be repaired, which can be modeled using a tree-like structure.

It is precisely the above-described behavior of avalanches that we exploit to obtain a cross-composition: The main gadgets select a Colorful Independent Set instance and independent-set vertices within that instance. Each such selection gadget has a trivial cluster-Π partition with one caveat: It has one (singleton) cluster too many in $G[A]$, and only this vertex can be pushed into the Π-side B. We call this vertex the *activator* vertex of the gadget. Pushing the activator vertex into B creates a forbidden induced subgraph for Π, requiring further pushes that propagate along a root-leaf path in a binary-tree-like structure. In the end, exactly one vertex corresponding to a leaf in this structure will be pushed from A to B, transmitting the choice to further gadgets.

Setup. Let t instances of Colorful Independent Set be given, with graphs $G_1, \dots, G_t$, respectively. Below, we use an instance and its index in $[t]$ interchangeably. Without loss of generality, assume that the following properties hold; they can be achieved by simple padding techniques. Each instance asks for an independent set of size k, each color class in each graph has n vertices and n as well as t are powers of two. In the following, let m be the maximum number of edges over all graphs G_i.

We construct an instance of Cluster-Π-Partition as described below. The instance consists of the graph G and asks for a cluster-Π partition (A, B) with at most d clusters in $G[A]$ (we specify d below). The graph G is constructed by first adding d vertices which we call *anchors* (see below). The clusters in any cluster-Π partition (A, B) of G with d clusters in $G[A]$ will extend these anchor vertices into larger cliques. We then successively add gadgets that are attached to these anchors. We first construct an instance-selection gadget that selects one of the given t instances. Then we add a vertex-selection gadget for each instance which selects k vertices in its corresponding instance if it has been selected. Finally, we add verification gadgets that ensure that the selected vertices are pairwise nonadjacent in the graph of the selected instance.

Throughout, we use the following notation. We denote by (A, B) an arbitrary fixed cluster-Π partition of G. We fix M to be a forbidden induced subgraph of Π with minimum number of vertices. By assumption, M contains at least three vertices. The vertices that we introduce will be in three disjoint categories: *helper* vertices, *dial* vertices, and *volatile* vertices. Their meaning is as follows. Helper vertices will always be contained in B and only serve to impose certain properties on other vertices. Dial vertices are normally in A and belong to a cluster extending around an anchor; some of these vertices may be pushed to B by an avalanche. On the other hand, volatile vertices are normally in B and may be pushed to A by an avalanche.

First, we introduce d anchor vertices, divided into $5 + 2k$ groups: a^1_1, a^1_2; $a^2_1, \dots, a^2_{2\log t}$; $a^3_1, \dots, a^3_{k+1}$; for each $i \in [k]$, $a^{3+i}_1, \dots, a^{3+i}_{\log n}$; for each $i \in [k]$, $a^{3+k+i}_1, \dots, a^{3+k+i}_n$; and $a^{5+2k}_1, \dots, a^{5+2k}_m$. Hence, we put $d := 2 + 2\log(t) + k + k\log n + kn + 2m$. The groups of anchors correspond to the gadgets constructed below in which they are used. Each anchor vertex is a dial vertex. We fix each of the anchors into A by introducing, for each anchor a^j_i, $d + 1$ copies of M and, for each copy, identifying an arbitrary vertex of that copy with a^j_i. The vertices different from a^j_i in the copies of M are helper vertices. If $a^j_i \in B$, then out of each of the d incident copies of M, at least one vertex is in A, and since these vertices are pairwise nonadjacent, $G[A]$ would contain at least $d + 1$ clusters, which is a contradiction. Thus, each anchor must be in A. When we construct cluster-Π partitions in the following we always tacitly assume that anchors are in A and all helper vertices are in B.

We associate each anchor a^j_i with a vertex set D^j_i that contains a^j_i and induces a clique in G (throughout the construction). We say that D^j_i is the *dial* of a^j_i. Initially, $D^j_i = \{a^j_i\}$. Later on, other vertices may *join* D^j_i; by saying a vertex v *joins* D^j_i, we mean that we put v into D^j_i and make v adjacent to all other vertices in D^j_i. Intuitively, the set of anchors corresponds to the clusters in $G[A]$. These clusters are divided into two types: Either an anchor's dial contains at least two vertices and the cluster consists only of vertices in the anchor's dial, or the anchor's dial contains only the anchor, and a single volatile vertex may join the anchor's cluster. We use the following notation.

▶ **Definition 3.2.** Let (A, B) be a cluster-Π partition for G and $\mathcal{D}$ be a set of dials. Partition (A, B) is *friendly* with respect to $\mathcal{D}$ if each singleton dial in $\mathcal{D}$ is a singleton cluster in $G[A]$.

Next, we introduce the operation of making three vertices exclusive. Intuitively, this operation is our main tool to fan out the possible pushes in avalanches according to a binary tree: When u is pushed to B, either v or w can be pushed to A to repair the partition. We use this construction extensively in the selection gadgets described below.

Given three vertices $u, v, w \in V(G)$, by *making u, v, and w exclusive* we mean: (i) introducing a copy of M into G, (ii) identifying three distinct vertices of M with u, v, and w, respectively, and (iii) fixing all remaining vertices of M (if any) into B by making each of them adjacent to both a_1^1 and a_2^1. The vertices in $V(M) \setminus \{u, v, w\}$ are helper vertices. Observe that $V(M) \setminus \{u, v, w\} \subseteq B$, because, otherwise, there would be a P_3 in $G[A]$ involving a_1^1 and a_2^1. Furthermore, not all three $u, v, w \in B$ since otherwise $G[B]$ contains a copy of M. When constructing cluster-Π partitions we will always tacitly assume that $V(M) \setminus \{u, v, w\} \subseteq B$ and ignore the vertices in $V(M) \setminus \{u, v, w\}$. Furthermore, to simplify showing that the constructed partition (A, B) is a cluster-Π partition we will show that $G[A]$ is a cluster graph, that $G[B] - \{u, v, w\} \in \Pi$, that at least one of u, v, w is in A and that $\{u, v, w\} \cap B$ do not have any neighbors in $G[B]$ other than $\{u, v, w\}$. Since Π is characterized by connected forbidden induced subgraphs and the helper vertices will not receive further neighbors, this suffices to prove that $G[B] \in \Pi$.

Instance Selection. The inner workings of the generic selection gadget described below use the necessary pushes along a binary-tree-like structure outlined above.

For use as an instance-selection gadget, we need to take special care so that the number of clusters used is roughly logarithmic in the number of instances. We achieve this by using only two clusters (represented by anchors and their dials) per level in the binary-tree-like structure of pushes. For use as a vertex-selection gadget, to bound the number of clusters in the size of the largest instance, we need to ensure that all the vertex-selection gadgets share their corresponding clusters. We achieve this by grouping the gadgets according to the groups of anchors above; each gadget uses only anchors in their corresponding group and shares these anchors with all other gadgets in this group. Essentially, the operation of vertices joining dials makes it possible to define the selection gadgets in a relatively local way.

We will use the following (generic) construction both for selecting an instance and for selecting the independent-set vertices in that instance. For this purpose, fix two construction parameters $p, q \in \mathbb{N}$, where p specifies which anchors (and dials) we use when constructing the gadget and q specifies how many possible choices shall be modeled. Herein, we require that q be a power of two. For the instance-selection gadget we will set $p = 2$ and $q = t$.

We introduce a new vertex v^*. Our goal is to construct a structure in which, starting from a trivial cluster-Π partition (A, B), putting $v^* \in B$ triggers an avalanche of pushes according to a path in a binary-tree-like structure. To this end, fix a rooted binary tree T with q leaves (corresponding to the $q = t$ instances of Colorful Independent Set for the instance-selection gadget). Say a vertex in T is on *level* $i \in [\log q]$ if its distance from the root is i. For $i \in [\log q]$, L_i denotes the set of vertices at level i. The tree T will not be part of the constructed graph, we use it only as a scaffold to define the actual vertices in the graph.

For each vertex $v \in V(T)$ except the root, introduce two vertices $\alpha(v), \beta(v)$ into G. Let i be the level of v. Connect $\alpha(v)$ to both a_{2i-1}^p and $\beta(v)$. Make $\beta(v)$ join D_{2i}^p. Furthermore, for each vertex $u \in L_i$, $i \in \{0, \ldots, \log q\}$, let v, w be the two children of u in T and make $\beta(u), \alpha(v), \alpha(w)$ exclusive. If $i = 0$, then let v, w be the two vertices in level 1 in T and make $v^*, \alpha(v), \alpha(w)$ exclusive instead. This completes the construction of the selection gadget. Vertex v^* is a volatile vertex, as is $\alpha(v)$ for $v \in V(T)$. Each $\beta(v)$, $v \in V(T)$, is a dial vertex. Call the constructed gadget $\mathsf{selection}(p, q)$, and say that v^* is the *activator vertex*, and that the vertices in $\{\beta(v) \mid v \in L_{\log q}\}$ are the *choice vertices*. We fix an arbitrary order of the choice vertices, so that we may speak of the ith choice vertex without confusion.

▶ **Lemma 3.3.** (♠) *Let G' be the graph before applying* selection(p, q) *and G the graph afterwards.*

(i) *If cluster-Π partition (A, B) has at most d clusters in $G[A]$ and the activator vertex is in B, then at least one choice vertex is in B.*

(ii) *If there is a cluster-Π partition (A', B') for G' with d clusters in $G'[A']$, then there is a cluster-Π partition (A, B) for G with $d + 1$ clusters, where the activator vertex is a singleton cluster and each choice vertex is in A. If (A', B') is friendly with respect to the dials D_i^p, then (A, B) is friendly with respect to the dials D_i^p.*

(iii) *If G' has a cluster-Π partition (A', B') that is friendly with respect to the dials D_i^p and such that $G'[A']$ contains at most d clusters, then, for each $i \in [q]$, there is a cluster-Π partition (A, B) of G, such that graph $G[A]$ contains at most d clusters, and out of all choice vertices only the ith one is in B (and, necessarily, the activator vertex is in B). Moreover, the choice vertex that is contained in B is isolated in $G[B]$.*

As mentioned, to construct the *instance-selection gadget*, we carry out selection$(2, t)$. For further reference, fix a bijection ϕ from the set of instances $[t]$ to the choice vertices produced by the construction. We use ϕ later to denote the choice vertex corresponding to an instance.

Vertex Selection. We now use the above construction selection$(\cdot, \cdot)$ to create vertex-selection gadgets for each instance and each color. Each vertex-selection gadget selects one vertex of the gadget's color into an independent set when activated by putting the activator vertex into B (which will be effected by the instance-selection gadget). The vertex-selection gadgets for each instance are distinct, but they use dials which are shared by all instances.

In the first part of the construction of the vertex-selection gadgets, for each instance $r \in [t]$ and color $i \in [k]$, carry out selection$(3 + i, n)$. Let $\psi_{r,i}^*$ be the corresponding activator vertex and fix a bijection $\psi_{r,i}$ from the vertices $V(G_r)$ of color i to the choice vertices. Make $\psi_{r,i}^*$ join D_{1+i}^3. Intuitively, if the activator vertex $\psi_{r,i}^*$ is put into B, the subgraph constructed by selection$(3 + i, n)$ enforces the push of a choice vertex into B, which by bijection $\psi_{r,i}$ correspond one-to-one to the vertices of color i in instance r. In this way, we model the selection of an independent-set vertex.

In the second part of the construction of the vertex-selection gadgets, we introduce a way to activate the vertex-selection gadgets of all colors if some instance $r \in [t]$ has been chosen. For this, carry out the following steps for each $r \in [t]$. Introduce two vertices u_r, v_r. Make $\phi(r)$, u_r, and v_r exclusive. Fix $u_r \in B$ by making it adjacent to both a_1, a_2. Make v_r adjacent to a_1^3 and, for each $i \in [k]$, make v_r adjacent to $\psi_{r,i}^*$. Vertex u_r is a helper vertex and v_r is a volatile vertex. This concludes the construction of the vertex-selection gadgets.

Intuitively, the selection of instance r is indicated by the fact that $\phi(r) \in B$. Since $u_r \in B$ and $\phi(r)$, u_r, and v_r are exclusive, $v_r \in A$. Vertex v_r forms a P_3 with a_1^3 and each $\psi_{r,i}^*$. Hence, the activator vertices $\psi_{r,i}^*$ of each vertex-selection gadget for instance r are in B. This enforces the selection of an independent-set vertex of each color.

By iteratively applying Lemma 3.3, we can show that the above-constructed graph has the properties that, if there is a cluster-Π partition (A, B) with d clusters in $G[A]$, then there is an instance for which the vertex-selection gadget for each color has one choice vertex in B (that is, the corresponding vertex is selected); and, vice-versa, for each possible selection of one vertex of each color in an instance, there is a corresponding cluster-Π partition.

Verification. For the verification gadgets it is again crucial to share clusters (anchors) between many gadgets to keep the overall number of clusters in A small. For this, we use $|V| = k \cdot n$ anchors that each represents, for each instance, one fixed vertex, and m pairs of anchors that each represents, for each instance, one fixed edge.

Due to space constraints, the details of the construction are not given here, but the working principle is as follows. Selecting a vertex v via a vertex-selection gadget will make it necessary to push a vertex corresponding to v into the cluster of its associated anchor. This push creates a P_3 in A for each incident edge e, necessitating a further push. Namely, we are required to push a vertex out of the cluster in A corresponding to one anchor associated with e. Pushing the corresponding vertex for the other endpoint of e into B will complete a forbidden induced subgraph, yielding that no two endpoints of an edge are selected. For the other direction of the correctness proof, we show that it is possible to configure the gadgets accordingly if one of the input instances is positive, which then concludes the proof of Theorem 3.1.

References

1 Faisal N. Abu-Khzam, Carl Feghali, and Haiko Müller. Partitioning a graph into disjoint cliques and a triangle-free graph. *Discrete Appl. Math.*, 190-191:1–12, 2015.

2 Demetrios Achlioptas. The complexity of G-free colourability. *Discrete Math.*, 165–166(0):21–30, 1997.

3 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014.

4 Marin Bougeret and Pascal Ochem. The complexity of partitioning into disjoint cliques and a triangle-free graph. *Discrete Appl. Math.*, 217:438–445, 2017.

5 Hajo Broersma, Fedor V. Fomin, Jaroslav Nešetřil, and Gerhard J. Woeginger. More about subcolorings. *Computing*, 69(3):187–203, 2002.

6 Sharon Bruckner, Falk Hüffner, and Christian Komusiewicz. A graph modification approach for finding core-periphery structures in protein interaction networks. *Algorithms Mol. Biol.*, 10:16, 2015.

7 Zh. A. Chernyak and A. A. Chernyak. About recognizing (α, β) classes of polar graphs. *Discrete Math.*, 62(2):133–138, 1986.

8 Ross Churchley and Jing Huang. On the polarity and monopolarity of graphs. *J. Graph Theory*, 76(2):138–148, 2014.

9 Ross Churchley and Jing Huang. Solving partition problems with colour-bipartitions. *Graph. Combinator.*, 30(2):353–364, 2014.

10 Marek Cygan, Fedor V Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

11 Reinhard Diestel. *Graph Theory, 4th Edition*. Springer, 2012.

12 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, Berlin, Heidelberg, 2013.

13 Alastair Farrugia. Vertex-partitioning into fixed additive induced-hereditary properties is NP-hard. *Electron. J. Comb.*, 11(1):R46, 2004.

14 Michael R. Fellows, Danny Hermelin, Frances Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009.

15 Jirí Fiala, Klaus Jansen, Van Bang Le, and Eike Seidel. Graph subcolorings: Complexity and algorithms. *SIAM J. Discrete Math.*, 16(4):635–650, 2003.

16 Stéphane Foldes and Peter L. Hammer. Split graphs. *Congr. Numer.*, 19:311–315, 1977.

17 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.

18 John Gimbel and Chris Hartman. Subcolorings and the subchromatic number of a graph. *Discrete Math.*, 272:139–154, 2003.

19 Pinar Heggernes, Dieter Kratsch, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Fixed-parameter algorithms for Cochromatic Number and Disjoint Rectangle Stabbing via iterative localization. *Infor. Comput.*, 231:109–116, 2013.

20 Iyad Kanj, Christian Komusiewicz, Manuel Sorge, and Erik Jan van Leeuwen. Parameterized algorithms for recognizing monopolar and 2-subcolorable graphs. *J. Comput. Syst. Sci.*, 92:22–47, 2018.

21 Sudeshna Kolay, Fahad Panolan, Venkatesh Raman, and Saket Saurabh. Parameterized Algorithms on Perfect Graphs for Deletion to (r, l)-Graphs. In *Proc. 41st MFCS*, volume 58 of *LIPIcs*, pages 75:1–75:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

22 Jan Kratochvíl and Ingo Schiermeyer. On the computational complexity of $(\mathcal{O}, \mathcal{P})$-partition problems. *Discuss. Math. Graph Theory*, 17(2):253–258, 1997.

23 Van Bang Le and Ragnar Nevries. Complexity and algorithms for recognizing polar and monopolar graphs. *Theor. Comput. Sci.*, 528:1–11, 2014.

24 Juraj Stacho. On 2-subcolourings of chordal graphs. In *Proc. 8th LATIN*, volume 4957 of *LNCS*, pages 544–554. Springer, 2008.

String Attractors: Verification and Optimization

Dominik Kempa
Department of Computer Science, University of Helsinki, Finland
dkempa@cs.helsinki.fi
https://orcid.org/0000-0003-2286-7417

Alberto Policriti
Department of Computer Science, University of Udine, Italy
alberto.policriti@uniud.it
https://orcid.org/0000-0001-8502-5896

Nicola Prezza
Department of Computer Science, University of Pisa, Italy
nicola.prezza@di.unipi.it
https://orcid.org/0000-0003-3553-4953

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

Abstract

String attractors [STOC 2018] are combinatorial objects recently introduced to unify all known dictionary compression techniques in a single theory. A set $\Gamma \subseteq [1..n]$ is a *k-attractor* for a string $S \in \Sigma^n$ if and only if every distinct substring of S of length at most k has an occurrence crossing at least one of the positions in Γ. Finding the smallest k-attractor is NP-hard for $k \geq 3$, but polylogarithmic approximations can be found using reductions from dictionary compressors. It is easy to reduce the k-attractor problem to a set-cover instance where the string's positions are interpreted as sets of substrings. The main result of this paper is a much more powerful reduction based on the truncated suffix tree. Our new characterization of the problem leads to more efficient algorithms for string attractors: we show how to check the validity and minimality of a k-attractor in near-optimal time and how to quickly compute exact solutions. For example, we prove that a minimum 3-attractor can be found in $\mathcal{O}(n)$ time when $|\Sigma| \in \mathcal{O}(\sqrt[3+\epsilon]{\log n})$ for some constant $\epsilon > 0$, despite the problem being NP-hard for large Σ.

2012 ACM Subject Classification Theory of computation → Data compression

Keywords and phrases Dictionary compression, String attractors, Set cover

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.52

Related Version A full version of the paper is available at [15], `https://arxiv.org/abs/1803.01695`.

1 Introduction

The goal of dictionary compression is to reduce the size of an input string by exploiting its repetitiveness. In the last decades, several dictionary compression techniques – some more powerful than others – were developed to achieve this goal: Straight-Line programs [17] (context-free grammars generating the string), Macro schemes [23] (a set of substring equations having the string as unique solution), the run-length Burrows-Wheeler transform [4] (a string permutation whose number of equal-letter runs decreases as the string's repetitiveness

© Dominik Kempa, Alberto Policriti, Nicola Prezza, and Eva Rotenberg;
licensed under Creative Commons License CC-BY

increases), and the compact directed acyclic word graph [3, 6] (the minimization of the suffix tree). Each scheme from this family comes with its own set of algorithms and data structures to perform compressed-computation operations – e.g. random access – on the compressed representation. Despite being apparently unrelated, in [16] all these compression schemes were proven to fall under a common general scheme: they all induce a set $\Gamma \subseteq [1..n]$ whose cardinality is bounded by the compressed representation's size and with the property that each distinct substring has an occurrence crossing at least one position in Γ. A set with this property is called a *string attractor*. Intuitively, positions in a string attractor capture "interesting" regions of the string; a string of low complexity (that is, more compressible), will generate a smaller attractor. Surprisingly, given such a set one can build a data structure of size $\mathcal{O}(|\Gamma| \operatorname{polylog}(n))$ supporting random access queries in optimal time [16]: string attractors therefore provide a universal framework for performing compressed computation on top of *any* dictionary compressor (and even optimally for particular queries such as random access).

These premises suggest that an algorithm computing the smallest string attractor for a given string would be a valuable tool for designing better compressed data structures. Unfortunately, computing a minimum string attractor is NP-hard. The problem remains NP-hard even under the restriction that only substrings of length at most k are captured by Γ, for any $k \geq 3$ and on large alphabets. In this case, we refer to the problem as *k-attractor*. Not all hope is lost, though: as shown in [16], dictionary compressors are actually heuristics for computing a small n-attractor (with polylogarithmic approximation rate w.r.t. the minimum), and, more generally, k-attractor admits a $\mathcal{O}(\log k)$-approximation based on a reduction to set cover. It is actually easy to find such a reduction: choose as universe the set of distinct substrings and as set collection the string's positions (i.e., set s_i contains all substrings crossing position i). The main limitation of this approach is that the set of distinct substrings could be quadratic in size; this makes the strategy of little use in cases where the goal is to design usable (i.e., as close as possible to linear-time) algorithms on string attractors.

The core result of this paper is a much more powerful reduction from k-attractor to set-cover: the universe $\mathcal{U}$ of our instance is equal to the set of edges of the k-truncated suffix tree, while the size of the set collection $\mathcal{S} \subseteq 2^{\mathcal{U}}$ is bounded by the size of the $(2k-1)$-truncated suffix tree. First of all, we obtain a universe that is always at least k times smaller than the naive approach. Moreover, the size of our set-cover instance does not depend on the string's length n, unless σ and k do. This allows us to show that k-attractor is actually solvable in polynomial time for small values of k and σ, and leads us to efficient algorithms for a wide range of different problems on string attractors.

The paper is organized as follows. In Section 1.1 we describe the notation used throughout the paper and we report the main notions related to k-attractors. In Section 1.2 we give the main theorem stating our reduction to set-cover (Theorem 5) and briefly discuss the results that we obtain in the rest of the paper by applying it. Theorem 5 itself is proven in Section 2 and is used in Section 3 to provide fast algorithms on string attractors. Finally, in Section 3.4 we introduce and study the complexity of the closely-related *sharp-k-attractor* problem: to capture all distinct substrings of length exactly k. The full version [15] of this paper covers additional material related to the approximation of minimum k-attractors.

1.1 Notation and definitions

Throughout we consider a string $S[1..n]$ of n symbols. We assume the reader to be familiar with the notions of *suffix tree* [24], *suffix array* [18], and *wavelet tree* [10, 21]. By $ST^k(S)$ we denote a k-truncated suffix tree of S, i.e., a compact trie containing all substrings of S

of length at most k. $\mathcal{E}(\mathcal{T})$ denotes the set of edges of the compact trie $\mathcal{T}$. $\mathcal{L}(\mathcal{T})$ denotes the set of leaves at maximum string depth of the compact trie $\mathcal{T}$ (i.e., leaves whose string depth is equal to the maximum string depth among all leaves). Let $e = \langle u, v\rangle$ be an edge in the (truncated) suffix tree of S. With $s(e)$ we denote the string read from the suffix tree root to (and including) the first character in the label of e. $\lambda(e) = |s(e)|$ is the length of this string. We will also refer to $\lambda(e)$ as the *string depth* of e. Note that edges $e_1, \dots, e_t$ of the k-truncated suffix tree have precisely the same labels $s(e_1), \dots, s(e_t)$ of the suffix tree edges $e'_1, \dots, e'_t$ at string depth $\lambda(e'_i) \le k$. It follows that we can use these two edge sets interchangeably when we are only interested in their labels (this will be the case in our results). Let $SA[1..n]$ denote the suffix array of S. $\langle l_e, r_e\rangle$, with $e \in \mathcal{E}(ST^k(S))$ being an edge in the k-truncated suffix tree, will denote the suffix array range corresponding to the string $s(e)$, i.e., $SA[l_e..r_e]$ contains all suffixes prefixed by $s(e)$.

Unless otherwise specified, we give the space of our data structures in words of $\Theta(\log n)$ bits each.

With the following definition we recall the notion of *k-attractor* of a string [16].

▶ **Definition 1.** A *k-attractor* of a string $S \in \Sigma^n$ is a set of positions $\Gamma \subseteq [1..n]$ such that every substring $S[i..j]$ with $i \le j < i + k$ has at least one occurrence $S[i'..j'] = S[i..j]$ with $j'' \in [i'..j']$ for some $j'' \in \Gamma$.

When $k = n$, we simply call Γ an *attractor* of S.

▶ **Definition 2.** A *minimal k-attractor* of a string $S \in \Sigma^n$ is a k-attractor Γ such that $\Gamma - \{j\}$ is not a k-attractor of S for any $j \in \Gamma$.

▶ **Definition 3.** A *minimum k-attractor* of a string $S \in \Sigma^n$ is a k-attractor Γ^* such that, for any k-attractor Γ of S, $|\Gamma^*| \le |\Gamma|$.

▶ **Theorem 4.** *[16, Thm. 4.2] The problem of deciding whether a string S admits a k-attractor of size at most t is NP-complete for $k \ge 3$.*

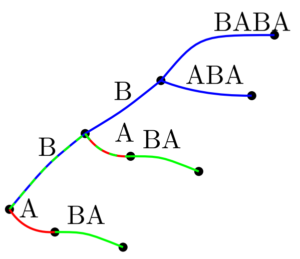
1.2 Overview of the contributions

Our main theorem is a reduction to set-cover based on the notion of truncated suffix tree:

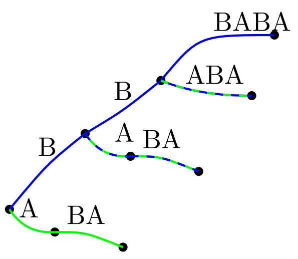
▶ **Theorem 5.** *Let $S' = \#^{k-1}S\#^{k-1}$, with $\# \notin \Sigma$. An instance of k-attractor can be reduced to a set-cover instance with universe equal to the set of edges of the k-truncated suffix tree of S and set collection with one element for each leaf of the $(2k-1)$-truncated suffix tree of S'.*

Figure 1 depicts the main technique (Lemma 11) standing at the core of our reduction: a set Γ is a valid attractor if and only if it marks (or colors, in the picture), all suffix tree edges.

Using the reduction of Theorem 5, we obtain the following results. First, we present efficient algorithms to check the validity and minimality of a k-attractor. Note that it is trivial to perform these checks in cubic time (or quadratic, with little more care). In Theorem 18 we show that we can check whether a set $\Gamma \subseteq [1..n]$ is a valid k-attractor for S in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ words of space. Using recent advances in compact data structures, we show how to further reduce this working space to $\mathcal{O}(n \log \sigma)$ bits without affecting query times when $k \le \sigma^{\mathcal{O}(1)}$, or with a small time penalty in the general case. In particular, when k is polynomial in the alphabet size, we can always check the correctness of a k-attractor in $\mathcal{O}(n)$ time and $\mathcal{O}(n \log \sigma)$ bits of space. With similar techniques, in Theorem 22 we show how to verify that Γ is a *minimal* k-attractor for S in near-optimal $\mathcal{O}(n \log n)$ time. To conclude, in Theorem 24 we show that a minimum k-attractor can be found in $\mathcal{O}(n) + \exp\big(\mathcal{O}(\sigma^k \log \sigma^k)\big)$ time. In particular, this result yields the following corollaries:



(a) The suffix tree of the string BBBABA, colored corresponding to attractor positions 2, 5, 6. This attractor is minimal: removing any position leaves some edge uncolored.



(b) The suffix tree of the string BBBABA, colored corresponding to attractor positions 3, 4. This attractor is minimum: it is minimal and of minimum size.

Figure 1 A position i "marks" (or, here, *colors*) a suffix tree edge e if and only if it crosses an occurrence of the string read from the root to the first letter in the label of e. A set of positions forms a k-attractor if and only if they color all edges of the k-truncated suffix tree (in this figure, $k = 6$ and we color the whole suffix tree). Dashed lines indicate that the edge has multiple colors. The string terminator \$ (and edges labeled with \$) is omitted for simplicity.

▶ **Corollary 6.** *k-attractor is in P when $\sigma^k \log \sigma^k \in \mathcal{O}(\log n)$.*

▶ **Corollary 7.** *For constant k, a minimum k-attractor can be found in $\mathcal{O}(n)$ time if there exists $\epsilon > 0$ such that $\sigma^{k+\epsilon} \in \mathcal{O}(\log n)$.*

Proof. Let $\sigma^{k+\epsilon} = \sigma^{k(1+\epsilon')}$, where $\epsilon' = \epsilon/k > 0$ is a constant. For any constant $\epsilon' > 0$ we have that $\sigma^k \log \sigma^k \in o(\sigma^k \cdot (\sigma^k)^{\epsilon'}) = o(\sigma^{k(1+\epsilon')})$. It follows that $\sigma^k \log \sigma^k \in o(\log n)$, i.e., by Theorem 24 we can find a minimum k-attractor in linear time. ◀

With the above result we can, for example, find a minimum 3-attractor in $\mathcal{O}(n)$ time when $\sigma \in \mathcal{O}(\sqrt[3+\epsilon]{\log n})$, for some $\epsilon > 0$ (keep in mind that 3-attractor is NP-complete for large alphabets).

2 A better reduction to set-cover

In this section we give our main result: a reduction from k-attractor to set-cover using a universe smaller than the one used in [16]. We start with an alternative characterization of k-attractors based on the k-truncated suffix tree.

▶ **Definition 8** (Marker). $j \in \Gamma$ is a *marker* for a suffix tree edge e if and only if

$$\exists i \in SA[l_e..r_e] \;:\; i \le j < i + \lambda(e)$$

Equivalently, we say that j *marks* e (see Figure 1).

▶ **Definition 9** (Edge marking). $\Gamma \subseteq [1..n]$ *marks* a suffix tree edge e if and only if there exists a $j \in \Gamma$ that marks e.

▶ **Definition 10** (Suffix tree k-marking). $\Gamma \subseteq [1..n]$ is a *suffix tree k-marking* if and only if it marks every edge e such that $\lambda(e) \le k$ (equivalently, every $e \in \mathcal{E}(ST^k(S))$).

When $k = n$ we simply say *suffix tree marking* (since all edges satisfy $\lambda(e) \le n$). We now show that the notions of k-attractor and suffix tree k-marking are equivalent.

▶ **Lemma 11.** *Γ is a k-attractor if and only if it is a suffix tree k-marking.*

Proof. ($\Rightarrow$) Let Γ be a k-attractor. Pick any suffix tree edge e such that $\lambda(e) \le k$. Then, $\lambda(e) = |s(e)| \le k$ and, by definition of k-attractor, there exists a $j \in \Gamma$ and an i such that $s(e) = S[i..i + |s(e)| - 1]$ and $i \le j \le i + |s(e)| - 1$. We also have that $i \in SA[l_e..r_e]$ (being $\langle l_e, r_e \rangle$ precisely the suffix array range of suffixes prefixed by $s(e)$). Putting these results together, we found an $i \in SA[l_e..r_e]$ such that $i \le j \le i + \lambda(e) - 1$ for some $j \in \Gamma$, which by Definition 9 means that Γ marks e. Since the argument works for any edge e at string depth at most k, we obtain that Γ is a suffix tree k-marking.

($\Leftarrow$) Let Γ be a suffix tree k-marking. Let, moreover, s be a substring of S of length at most k. Consider the lowest suffix tree edge e (i.e., the e with maximum $\lambda(e)$) such that $s(e)$ prefixes s. In particular, $\lambda(e) \le k$. Note that, by definition of suffix tree, every occurrence $S[i..i + |s(e)| - 1] = s(e)$ of $s(e)$ in S prefixes an occurrence of s: $S[i..i + |s| - 1] = s$. By definition of k-marking, there exists a $j \in \Gamma$ such that j is a marker for e, which means (by Definition 8) that $\exists i \in SA[l_e..r_e] \ : \ i \le j < i + \lambda(e)$. Since $i \in SA[l_e..r_e]$, $SA[i]$ is an occurrence of $s(e)$, and therefore of s. But then, we have that $i \le j < i + \lambda(e) = i + |s(e)| \le i + |s|$, i.e., $S[SA[i]..SA[i] + |s| - 1]$ is an occurrence of s crossing $j \in \Gamma$. Since the argument works for every substring s of S of length at most k, we obtain that Γ is a k-attractor. ◀

An equivalent formulation of Lemma 11 is that Γ is a k-attractor if and only if it marks all edges of the k-truncated suffix tree. In particular (case $k = n$), Γ is an attractor if and only if it is a suffix tree marking.

Lemma 11 will be used to obtain a smaller universe $\mathcal{U}$ in our set-cover reduction. With the following lemmas we show that also the size of the set collection $\mathcal{S}$ can be considerably reduced when k and σ are small.

▶ **Definition 12** (k-equivalence)**.** Two positions $i, j \in [1..n]$ are k-equivalent, indicated as $i \equiv_k j$, if and only if

$$S'[i - k + 1..i + k - 1] = S'[j - k + 1..j + k - 1]$$

where $S'[i] = \#$ if $i < 1$ or $i > n$ (note that we allow negative positions) and $S'[i] = S[i]$ otherwise, and $\# \notin \Sigma$ is a new character.

It is easy to see that k-equivalence is an equivalence relation. First, we bound the size of the distinct equivalence classes of $\equiv_k$ (i.e., the size of the quotient set $[1..n]/\equiv_k$).

▶ **Lemma 13.** $|[1..n]/\equiv_k| = |\mathcal{L}(ST^{2k-1}(S'))| \le \min\{n, \sigma^{2k-1} + 2k - 2\}$

Proof. By definition of $\equiv_k$, the set $[1..n]/\equiv_k$ has one element per distinct substring of length $(2k - 1)$ in S', that is, per distinct path from the suffix tree root to each of the nodes in $\mathcal{L}(ST^{2k-1}(S'))$. Clearly, $|\mathcal{L}(ST^{2k-1}(S'))| \le n$. On the other hand, there are at most σ^{2k-1} distinct substrings of length $2k - 1$ on Σ. Moreover, there are $2k - 2$ additional substrings to consider on the borders of S' (to include the runs of symbol $\#$). It follows that the cardinality of $\mathcal{L}(ST^{2k-1}(S'))$ is upper-bounded also by $\sigma^{2k-1} + 2k - 2$. ◀

We now show that any minimal k-attractor can have at most one element from each equivalence class of $\equiv_k$.

▶ **Lemma 14.** *If Γ is a minimal k-attractor, then for any $1 \le i \le n$ it holds $|\Gamma \cap [i]_{\equiv_k}| \le 1$.*

Proof. Suppose, by contradiction, that $|\Gamma \cap [i]_{\equiv_k}| > 1$ for some i. Then, let $j, j' \in \Gamma \cap [i]_{\equiv_k}$, with $j \ne j'$. By definition of $\equiv_k$, $S'[j - k + 1..j + k - 1] = S'[j' - k + 1..j' + k - 1]$. This means that if a substring of S of length at most k has an occurrence crossing position j

in Γ then it has also one occurrence crossing position $j' \in (\Gamma - \{j\})$. On the other hand, any other substring occurrence crossing any position $j'' \neq j, j'$ is also captured by $\Gamma - \{j\}$ since j'' belongs to this set. This implies that $\Gamma - \{j\}$ is a k-attractor, which contradicts the minimality of Γ. ◀

Moreover, if we swap any element of a k-attractor with an equivalent element then the resulting set is still a k-attractor:

▶ **Lemma 15.** *Let Γ be a k-attractor. Then, $(\Gamma - \{j\}) \cup \{j'\}$ is a k-attractor for any $j \in \Gamma$ and any $j' \equiv_k j$.*

Proof. Pick any occurrence of a substring s, $|s| \leq k$, crossing position j. By definition of $\equiv_k$, since $j' \equiv_k j$ there is also an occurrence of s crossing j'. This implies that $\Gamma' = (\Gamma - \{j\}) \cup \{j'\}$ is a k-attractor. ◀

Lemmas 14 and 15 imply that we can reduce the set of candidate positions from $[1..n]$ to $\mathcal{C} = \{\min(I) \mid I \in [1..n]/\equiv_k\}$ (that is, an arbitrary representative – in this case, the minimum – from any class of $\equiv_k$), and still be able to find a minimal/minimum k-attractor. Note that, by Lemma 13, $|[1..n]/\equiv_k| \leq \min\{n, \sigma^{2k-1} + 2k - 2\}$.

We can now prove our main theorem.

Proof of Theorem 5. We build our set-cover instance $\langle \mathcal{U}, \mathcal{S} \rangle$ as follows. We choose $\mathcal{U} = \mathcal{E}(ST^k(S))$, i.e., the set of edges of the k-truncated suffix tree. The set collection $\mathcal{S}$ is defined as follows: let $s_i = \{e \in \mathcal{E}(ST^k(S)) \mid i \text{ marks } e\}$, $\mathcal{C} = \{\min(I) \mid I \in [1..n]/\equiv_k\}$ and put:

$$\mathcal{S} = \{s_i \mid i \in \mathcal{C}\}.$$

By definition of $\equiv_k$, each $I \in [1..n]/\equiv_k$ is unambiguously identified by a substring of length $2k-1$ of the string $S' = \#^{k-1}S\#^{k-1}$. We therefore obtain $|\mathcal{S}| = |\mathcal{L}(ST^{2k-1}(S'))|$. We now prove the correctness and completeness of the reduction.

Correctness. By the definition of our reduction, a solution $\{s_{i_1}, \dots, s_{i_\gamma}\}$ to $\langle \mathcal{U}, \mathcal{S} \rangle$ yields a set $\Gamma = \{i_1, \dots, i_\gamma\}$ of positions marking every edge in $\mathcal{E}(ST^k(S))$. Then, Lemma 11 implies that Γ is a k-attractor.

Completeness. Let $\Gamma = \{i_1, \dots, i_\gamma\}$ be a minimal k-attractor. Then, Lemmas 14 and 15 imply that the following set is also a minimal k-attractor of the same size: $\Gamma' = \{j_1 = \min([i_1]_{\equiv_k}), \dots, j_\gamma = \min([i_\gamma]_{\equiv_k})\}$. Note that $\Gamma' \subseteq \{\min(I) \mid I \in [1..n]/\equiv_k\}$. By Lemma 11, Γ' marks every edge in $\mathcal{E}(ST^k(S))$. Then, by definition of our reduction the collection $\{s_{j_1}, \dots, s_{j_\gamma}\}$ covers $\mathcal{U} = \mathcal{E}(ST^k(S))$. ◀

In the rest of the paper, we use the notation $\mathcal{U} = \mathcal{E}(ST^k(S))$ and $\mathcal{C} = \{\min(I) \mid I \in [1..n]/\equiv_k\}$ to denote the universe to be covered (edges of the k-truncated suffix tree) and the candidate attractor positions, respectively. Recall, moreover, that $|\mathcal{U}| \leq \min\{n, \sigma^k\}$ and $|\mathcal{C}| \leq \min\{n, \sigma^{2k-1} + 2k - 2\}$.

3 Faster algorithms

In this section we use properties of our reduction to provide faster algorithms for a range of problems: (i) checking that a given set $\Gamma \subseteq [1..n]$ is a k-attractor, (ii) checking that a given set is a *minimal* k-attractor, and (iii) finding a minimum k-attractor. We note that problems (i)-(ii) admit naive cubic solutions, while problem (iii) is NP-hard for $k \geq 3$ [16].

We assume that the input string S is over the integers alphabet $[1..\sigma]$ where $\sigma \in n^{\mathcal{O}(1)}$. Note that suffix-sorting can be performed in linear time and space on such alphabets [14].

3.1 Checking the attractor property

Given a string S, a set $\Gamma \subseteq [1..n]$, and an integer $k \geq 1$, is Γ a k-attractor for S? We show that this question can be answered in $\mathcal{O}(n)$ time.

The main idea is to use Lemma 11 and check, for every suffix tree edge e at string depth at most k, if Γ marks e. Consider the suffix array $SA[1..n]$ of S and the array $D[1..n]$ defined as follows: $D[i] = \mathrm{succ}(\Gamma, SA[i]) - SA[i]$, where $\mathrm{succ}(X, x)$ returns the smallest element larger than or equal to x in the set X (i.e., $D[i]$ is the distance between $SA[i]$ and the element of Γ following – and possibly equal to – $SA[i]$). D can be built in linear time and space by creating a bit-vector $B[1, n]$ such that $B[i] = 1$ iff $i \in \Gamma$ and pre-processing B for constant-time successor queries [13, 5]. We build a range-minimum data structure (RMQ) on D ($\mathcal{O}(n)$ bits of space, constant query time [9]). Then for every suffix tree edge e such that $\lambda(e) \leq k$, we check (in constant time) that $\lambda(e) > \min(D[l_e..r_e])$. The following lemma ensures that this is equivalent to checking whether Γ marks e.

▶ **Lemma 16.** *$\lambda(e) > \min(D[l_e..r_e])$ if and only if Γ marks e.*

Proof. ($\Rightarrow$) Assume that $\lambda(e) > \min(D[l_e..r_e])$. By definition of D, this means that there exist an index $i' \in [l_e..r_e]$ and a $j \in \Gamma$, with $j \geq i = SA[i']$, such that $j - i = D[i'] < \lambda(e)$. Equivalently, $i \leq j < i + \lambda(e)$, i.e., Γ marks e.

($\Leftarrow$) Assume that Γ marks e. Then, by definition, there exist an index $i' \in [l_e..r_e]$ and a $j \in \Gamma$ such that $SA[i'] = i \leq j < i + \lambda(e)$. Then, $j - i < \lambda(e)$. Since $D[i']$ is computed taking the $j \in \Gamma$, $j \geq SA[i']$, minimizing $j - SA[i']$, it must be the case that $D[i'] \leq j - i < \lambda(e)$. Since $i' \in [l_e..r_e]$, this implies that $\min(D[l_e..r_e]) < \lambda(e)$. ◀

Together, Lemmas 11 and 16 imply that, if $\lambda(e) > \min(D[l_e..r_e])$ for every edge at string depth at most k, then Γ is a k-attractor for S. Since the suffix tree, as well as the other structures used by our algorithm, can be built in linear time and space on alphabet $[1..n]$ [7] and checking each edge takes constant time, we obtain that the problem of checking whether a set $\Gamma \subseteq [1..n]$ is a valid k-attractor can be solved in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ words of space. We now show how to improve upon this working space by using recent results in the field of compact data structures. In the following result, we assume that the input string is packed in $\mathcal{O}(n \log \sigma)$ bits (that is, $\mathcal{O}(n/\log_\sigma n)$ words).

We first need the following Lemma from [2]:

▶ **Lemma 17.** *[2, Thm. 3] In $\mathcal{O}(n)$ time and $\mathcal{O}(n \log \sigma)$ bits of space we can enumerate the following information for each suffix tree edge e:*

- *The suffix array range $\langle l_e, r_e \rangle$ of the string $s(e)$, and*
- *the length $\lambda(e)$ of $s(e)$.*

Proof. In [2, Thm. 3] (see also [1]) the authors show how to enumerate the following information for each right-maximal substring W of S in $\mathcal{O}(n)$ time and $\mathcal{O}(n \log \sigma)$ bits of space: $|W|$ and the suffix array range $\mathit{range}(Wb)$ of the string Wb, for all $b \in \Sigma$ such that Wb is a substring of S. Since W is right-maximal, those Wb are equal to our strings $s(e)$ (for every edge e). It follows that our problem is solved by outputting all $\mathit{range}(Wb)$ and $|Wb|$ returned by the algorithm in [2, Thm. 3]. ◀

We can now prove our theorem. Note that the input set $\Gamma \subseteq [1..n]$ can be encoded in n bits, so also the input fits in $\mathcal{O}(n \log \sigma)$ bits.

▶ **Theorem 18.** *Given a string $S \in [1..\sigma]^n$, a set $\Gamma \subseteq [1..n]$, and an integer $k \geq 1$, we can check whether Γ is a k-attractor for S in:*

- *Optimal $\mathcal{O}(n \log \sigma)$ bits of space and $\mathcal{O}(n \log^{\epsilon} n)$ time, for any constant $\epsilon > 0$, or*
- *$\mathcal{O}(n(\log \sigma + \log k))$ bits of space and $\mathcal{O}(n)$ time.*

Proof. To achieve the first trade-off we will replace the D array (occupying $\mathcal{O}(n \log n)$ bits) with a smaller data structure supporting random access to D. We start by replacing the standard suffix array with a compressed suffix array (CSA) [8, 11]. Given a text stored in $\mathcal{O}(n \log \sigma)$ bits, the CSA can be built in deterministic $\mathcal{O}(n)$ time and optimal $\mathcal{O}(n \log \sigma)$ bits of space [20], and supports access queries to the suffix array SA in $\mathcal{O}(\log^{\epsilon} n)$ time [11], for any constant $\epsilon > 0$ chosen at construction time. Given that $D[i] = \text{succ}(\Gamma, SA[i]) - SA[i]$ and we can compute the successor function in constant time using a $\mathcal{O}(n)$-bit data structure (array B), $D[i]$ can be computed in $\mathcal{O}(\log^{\epsilon} n)$ time. Using access to D, the RMQ data structure (occupying $\mathcal{O}(n)$ bits) can be built in $\mathcal{O}(n \log^{\epsilon} n)$ time and $\mathcal{O}(n)$ bits of space [9, Thm. 5.8]. At this point, observe that the order in which we visit suffix tree edges does not affect the correctness of our algorithm. By using Lemma 17 we can enumerate $\lambda(e)$ and $\langle l_e, r_e \rangle$ for every suffix tree edge e in linear time and compact space, and check $\lambda(e) > \min(D[l_e..r_e])$, whenever $\lambda(e) \leq k$ (Lemma 16).

To achieve the second trade-off we observe that in our algorithm we only explore the suffix tree up to depth k (i.e., we only perform the check of Lemma 16 when $\lambda(e) \leq k$), hence any $D[i] > k$ can be replaced with $D[i] = k + 1$ without affecting the correctness of the verification procedure. In this way, array D can be stored in just $\mathcal{O}(n \log k)$ bits. To compute the D array in $\mathcal{O}(n)$ time and compact space we observe that it suffices to access all pairs $\langle i, SA[i] \rangle$ in *any* order (not necessarily $\langle 1, SA[1] \rangle, \langle 2, SA[2] \rangle, \dots$). From [2, Thm. 10], in $\mathcal{O}(n)$ time and $\mathcal{O}(n \log \sigma)$ bits of space we can build a compressed suffix array supporting constant-time LF function computation. By repeatedly applying LF from the first suffix array position, we enumerate entries of the inverse suffix array ISA in right-to-left order in $\mathcal{O}(n)$ time [2, Lem. 1]. This yields the sequence of pairs $\langle ISA[i], i \rangle = \langle j, SA[j] \rangle$, for $i = n, \dots, 1$ and $j = ISA[i]$, which can be used to compute D in linear time and compact space. As in the first trade-off, we use Lemma 17 to enumerate $\lambda(e)$ and $\langle l_e, r_e \rangle$ for every suffix tree edge e, and check $\lambda(e) > \min(D[l_e..r_e])$, whenever $\lambda(e) \leq k$ (Lemma 16). ◀

Note that with the second trade-off of Theorem 18 we achieve $\mathcal{O}(n)$ time and optimal $\mathcal{O}(n \log \sigma)$ bits of space when $k \leq \sigma^{\mathcal{O}(1)}$ (in particular, this is always the case when k is constant). Note also that, since we now assume that the input string is packed in $\mathcal{O}(n / \log_{\sigma} n)$ words, the running time is not optimal (as $\Omega(n / \log_{\sigma} n)$ is a lower-bound in this model).

3.2 Checking minimality

Given a string S, a set $\Gamma \subseteq [1..n]$, and an integer $k \geq 1$, is Γ a *minimal* k-attractor for S? The main result of this section is that this question can be answered in near-optimal $\mathcal{O}(n \log n)$ time.

We first show that minimal k-attractors admit a convenient characterization based on the concept of suffix tree k-marking.

▶ **Definition 19** (k-necessary position). $j \in \Gamma$ is *k-necessary* with respect to Γ if and only if there is at least one suffix tree edge e such that:

1. $\lambda(e) \leq k$,
2. j marks e, and
3. If $j' \in \Gamma$ marks e, then $j' = j$

▶ **Definition 20** (k-necessary set)**.** Γ is *k-necessary* if and only if all its elements are k-necessary with respect to Γ.

When Γ is clear from the context, we just say *k-necessary* (referring to some $j \in \Gamma$) instead of *k-necessary with respect to* Γ.

▶ **Lemma 21.** *Γ is a minimal k-attractor if and only if:*
1. *It is a k-attractor, and*
2. *it is k-necessary.*

Proof. ($\Rightarrow$) Let Γ be a minimal k-attractor. Let $j \in \Gamma$. Since Γ is minimal, $\Gamma - \{j\}$ is not a k-attractor. From Theorem 11, this implies that $\Gamma - \{j\}$ is not a k-marking, i.e., there exists a suffix tree edge e, with $\lambda(e) \leq k$, that is not marked by $\Gamma - \{j\}$. On the other hand, the fact that Γ is a k-attractor implies (Theorem 11) that Γ is a k-marking, i.e., it also marks edge e. This, in particular, implies that j marks e. Now, let $j' \in \Gamma$ be a position that marks e. Assume, by contradiction, that $j' \neq j$. Then, $j' \in \Gamma - \{j\}$, which implies that $\Gamma - \{j\}$ marks e. This is a contradiction, therefore it must be the case that $j' = j$, i.e., j is k-necessary. Since the argument works for any $j \in \Gamma$, we obtain that all $j \in \Gamma$ are k-necessary.

($\Leftarrow$) Assume that Γ is a k-attractor and all $j \in \Gamma$ are k-necessary. Then, choose an arbitrary $j \in \Gamma$. By Definition 19, there exists an edge e that is only marked by j, i.e., for every $j' \in \Gamma - \{j\}$, j' does not mark e. This implies (Theorem 11) that $\Gamma - \{j\}$ is not a k-attractor. Since the argument works for any $j \in \Gamma$, we obtain that Γ is a minimal k-attractor. ◀

A naive solution for the minimality-checking problem is to test the k-attractor property on $\Gamma - \{i\}$ for every $i \in \Gamma$ using Theorem 18. This solution, however, runs in quadratic time. Our efficient strategy relies on colored range reporting and consists in checking, for every suffix tree edge e, if there is only one $j \in \Gamma$ marking it. In this case, we flag j as necessary. If, in the end, all attractor positions are flagged as necessary, then the attractor is minimal by Lemma 21. Notice that it can turn out that a single suffix tree edge can be marked by more than one necessary $j \in \Gamma$.

▶ **Theorem 22.** *Given a string $S \in [1..\sigma]^n$, a set $\Gamma \subseteq [1..n]$, and an integer $k \geq 1$, we can check whether Γ is a minimal k-attractor for S in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n \log |\Gamma|)$ space.*

Proof. We associate to each element in Γ a distinct color from the set $\Sigma_\Gamma = \{c_i \mid i \in \Gamma\}$, and we build a two-dimensional Σ_Γ-colored grid $L \subseteq [1..n]^2 \times \Sigma_\Gamma$ (i.e., each point $\langle i, j\rangle$ in L is associated with a color from Σ_Γ) defined as $L = \{\langle i, D[i], c_{SA[i]+D[i]}\rangle,\ i = 1, ..., n\}$, that is, at coordinates $\langle i, D[i]\rangle$ we insert a point "colored" with the color associated to the attractor position immediately following – and possibly equal to – $SA[i]$. Then, for every suffix tree edge e we check that $L \cap [l_e..r_e] \times [0..\lambda(e)-1]$ contains at least two distinct colors. If there are at least two distinct colors $c_k \neq c_{k'}$ in the range $[l_e..r_e] \times [0..\lambda(e) - 1]$, then we do not mark k and k' as necessary (note that they could be marked later by some other edge, though). However, even if there is only one color c_k in the range this may not be enough to mark k as necessary. The reason for this is that in array D we are tracking only the attractor position i' *immediately following* each text position i; it could well be that the attractor position $i'' > i'$ immediately following i' marks e, but we miss it because we track only i'. This problem can be easily solved inserting in L also a point corresponding to the *second* nearest attractor position following every text position (so the number of points only doubles). It is easy to see that this is sufficient to solve our problem, since we only aim at enumerating at most two distinct colors in a range.

At this point, we have reduced the problem to the so-called *three-sided colored orthogonal range reporting* problem in two dimensions: report the distinct colors inside a three-sided orthogonal range in a grid. For this problem, the fastest known data structure takes $\mathcal{O}(n \log n)$ space and answers queries in $\mathcal{O}(\log^2 n + i)$ time, where n is the number of points in the grid and i is the number of returned points [12]. This would result in an overall running time of $\mathcal{O}(n \log^2 n)$ for our algorithm. We note that our problem is, however, simpler than the general one. In our case, it is enough to list *two* distinct colors (if any); we are not interested in counting the total number of such colors or reporting an arbitrary number of them.

Our solution relies on wavelet trees [10]. First, we pre-process the set of $v \leq 2n$ points so that they fit in a grid $[1..v] \times [1..v]$ such that every row and every column contain exactly one point. Mapping the original query on this grid can be easily done in constant time using well-established rank reduction techniques that we do not discuss here (see, e.g. [22]). We can view this grid as an integer vector $V \in [1..v]^v$, where each $V[i]$ is associated with a color $V[i].c \in \Sigma_\Gamma$. We build a wavelet tree $WT(V)$ on V. Let u_s denote the internal node of $WT(V)$ reached following the path $s \in \{0,1\}^*$. With V_s we denote the subsequence of V associated with u_s, i.e., the subsequence of V such that the binary representation of each $V_s[i]$ is prefixed by s. For each internal node u_s we store (i) the sequence of colors $C_s = V_s[1].c, \dots, V_s[|V_s|].c$, and (ii) a bitvector $B_s[1..|V_s|]$ such that $B_s[1] = 0$ and $B_s[i] \neq B_s[i-1]$ iff $C_s[i] \neq C_s[i-1]$. We pre-process each B_s for constant-time rank and select queries [13, 5]. Overall, our data structure takes $\mathcal{O}(n \log \Gamma)$ words of space (that is, $\mathcal{O}(n \log \Gamma \log n)$ bits: at each of the $\log n$ levels of the wavelet tree we store $v \leq 2n$ colors).

To report two distinct colors in the range $[l, r] \times [l', r']$, we find in $\mathcal{O}(\log v)$ time the $\mathcal{O}(\log v)$ nodes of $WT(V)$ covering $[l', r']$ as usually done when solving orthogonal range queries on wavelet trees (see [21] for full details). For each such node u_s, let the range $V_s[l_s, r_s]$ contain the elements in common between V_s and $V[l..r]$ (i.e., the range obtained mapping $[l..r]$ on V_s). We check whether in $B_s[l_s..r_s]$ there are two distinct bits at adjacent positions. If this is the case, we locate their positions $B_s[i_0] = 0$ and $B_s[i_1] = 1$, with $l_s \leq i_0 = i_1 - 1 \leq r_s$ (the case $i_1 = i_0 - 1$ is symmetric), and return the colors $C_s[i_0]$ and $C_s[i_1]$. By construction of B_s, $C_s[i_0] \neq C_s[i_1]$ and therefore we are done. Note that i_0 and i_1 can be easily found in constant time using rank/select queries on B_s.

If, on the other hand, all sequences $B_s[l_s..r_s]$ are unary, then we just need to retrieve the $\mathcal{O}(\log v)$ colors $C_s[l_s]$, for all the u_s covering the interval $[l', r']$, and check if any two of them are distinct (e.g. radix-sort them in linear time). Also this step runs in $\mathcal{O}(\log v)$ time.

Overall, our solution uses $\mathcal{O}(n \log |\Gamma|)$ space and runs in $\mathcal{O}(n \log v) = \mathcal{O}(n \log n)$ time. ◀

3.3 Computing a minimum k-attractor

Computing a minimum k-attractor is NP-hard for $k \geq 3$ and large σ. In this section we show that the problem is actually polynomial-time-solvable for small k and σ. Our algorithm takes advantage of both our reduction to set-cover and the verification algorithm of Theorem 18.

First, we give an upper-bound to the cardinality of the set of all minimal k-attractors. This will speed up our procedure for finding a minimum k-attractor (which is, in particular, minimal). By Lemma 13, there are no more than $\exp\big(\mathcal{O}(\sigma^{2k})\big)$ k-attractors for S. With the following lemma, we give a better upper-bound to the number of *minimal* k-attractors.

▶ **Lemma 23.** *The number of minimal k-attractors is* $\exp\big(\mathcal{O}(\sigma^k \log \sigma^k)\big)$.

Proof. Let minimal(σ, k) denote the maximum number of minimal k-attractors on the alphabet $[1..\sigma]$ (independently of the string length n). Let Γ be a minimal k-attractor. By Lemma 21, for every $j \in \Gamma$ there is at least one edge $e \in \mathcal{U}$ marked by j only. Let edge : $\Gamma \rightarrow \mathcal{U}$ be the function defined as follows: edge$(j) = e$ such that (i) e is marked by j only, and (ii) among all edges marked by j only, e is the one with the lexicographically smallest $s(e)$ (where, if $s(e')$ prefixes $s(e'')$, then we consider $s(e')$ smaller than $s(e'')$ in lexicographic order). Let $\mathcal{U}' = \text{edge}(\Gamma)$ be the image of Γ through edge function. By its definition, edge is a bijection between Γ and $\mathcal{U}'$. This implies that $|\Gamma| = |\mathcal{U}'| \le |\mathcal{U}| \le \sigma^k$: a minimal k-attractor is a set of cardinality at most σ^k chosen from a universe $\mathcal{C}$ of size at most $|\mathcal{C}| \le \sigma^{2k-1} + 2k - 2 \le \sigma^{2k}$, therefore: minimal$(\sigma, k) \le \sum_{i=1}^{\sigma^k} \binom{\sigma^{2k}}{i}$. We now give an upper-bound to the function $f(N,t) = \sum_{i=1}^{t} \binom{N}{i}$, where we assume $t \ge 2$ for simplicity (the hypothesis holds in our case since $t = \sigma^k$). Then, we will plug our bound in the above inequality. Since $\binom{N}{i} < \frac{N^i}{i!}$, we have that

$$\begin{aligned} f(N,t) &< \textstyle\sum_{i=1}^{t} \frac{N^i}{i!} \\ &= \textstyle\sum_{i=1}^{t} \left(\frac{N}{t}\right)^i \cdot \frac{t^i}{i!} \\ &\le \textstyle\sum_{i=1}^{t} \left(\frac{N}{t}\right)^i \cdot \sum_{i=1}^{t} \frac{t^i}{i!} \\ &\in \mathcal{O}\left(\left(\frac{N\cdot e}{t}\right)^t\right) \end{aligned}$$

We obtain our claim: minimal$(\sigma, k) \le f(\sigma^{2k}, \sigma^k) \in \mathcal{O}(e^{\sigma^k} \cdot \sigma^{k\sigma^k}) \le \exp\big(\mathcal{O}(\sigma^k \log \sigma^k)\big)$. ◀

Using the above lemma, we now provide a strategy to find a minimum k-attractor.

▶ **Theorem 24.** *A minimum k-attractor can be found in $\mathcal{O}(n) + \exp\big(\mathcal{O}(\sigma^k \log \sigma^k)\big)$ time.*

Proof. Let $c(i) = S'[i-k+1..i+k-1]$, where $S'[i] = \# \notin \Sigma$ if $i < 1$ or $i > n$ and $S'[i] = S[i]$ otherwise, be the context string associated to position i. Consider the string

$$C = c(i_1)\$c(i_2)\$\ldots\$c(i_t)$$

where $\{i_1, i_2, \ldots, i_t\} = \mathcal{C}$ and $\# \ne \$ \notin \Sigma$. By our choice of $\mathcal{C}$, the length of this string is $|C| = (|\mathcal{C}| \cdot 2k) - 1 \le (\sigma^{2k-1} + 2k) \cdot 2k \in \mathcal{O}(\sigma^{2k} \cdot k) \le \exp\big(\mathcal{O}(\log \sigma^k)\big)$. We can build C in $\mathcal{O}(n + |C|)$ time using the suffix tree of S (i.e., extracting all paths from the root to nodes at string depth at most $2k-1$).

Let now $\Gamma'' \subseteq \{k \cdot (2j+1) \mid j = 0, \ldots, t-1\}$. It is easy to see that $\Gamma' = \{i \mid C[i] = \$ \textit{ or } C[i] = \#\} \cup \Gamma''$ is a k-attractor for C if and only if the set $\Gamma = \{i_{(x-k)/(2k)} \mid x \in \Gamma''\}$ is a k-attractor for S. Suppose that Γ' is a k-attractor for C, and consider a substring s of S of length at most k. By construction of C, s is also a substring of C; in particular, there is an occurrence $C[i..i+|s|-1] = s$ crossing a position $k \cdot (2j+1) \in \Gamma''$, for some j. Then, $i_j \in \Gamma$ and, by the way we defined C, there is an occurrence of s crossing position i_j in S. Conversely, suppose that Γ is a k-attractor for S, and let s be a substring of C of length at most k. If s contains either \$ or #, then it must cross one of the positions in $\{i \mid C[i] = \$ \textit{ or } C[i] = \#\} \subseteq \Gamma'$. Otherwise, it appears inside one of the substrings $c(i_k)$, for some $k \in [1..t]$. But then, this means that s appears in S and, in particular, that it has some occurrence $s' = s$ crossing a position $i_j \in \Gamma$. From the way we constructed C, s has an occurrence in C crossing position $k \cdot (2j+1) \in \Gamma'' \subseteq \Gamma'$.

At this point, we check whether Γ' is a k-attractor for C for all possible Γ'', and return the smallest such set. Instead of trying all subsets of $\mathcal{C}$, we use Lemma 23 and generate only subsets of $\mathcal{C}$ of size at most σ^k; these subsets will include all minimal k-attractors and, in particular, all minimum k-attractors. By Lemma 23, there are at most $\exp\big(\mathcal{O}(\sigma^k \log \sigma^k)\big)$ such sets, and each verification takes linear $\mathcal{O}(|C|) = \exp\big(\mathcal{O}(\log \sigma^k)\big)$ time using Theorem 18. Overall, our algorithm for the minimum k-attractor runs in $\mathcal{O}(n) + \exp\big(\mathcal{O}(\log \sigma^k)\big) \cdot \exp\big(\mathcal{O}(\sigma^k \log \sigma^k)\big)$ time, which simplifies to $\mathcal{O}(n) + \exp\big(\mathcal{O}(\sigma^k \log \sigma^k)\big)$. ◀

3.4 Sharp attractors

The complexity of k-attractor has been fully characterized in [16], except for the particular case $k = 2$: for $k \geq 3$, the problem has been proven to be NP-complete, while for $k = 1$ it is clearly solvable in linear time. While we have not been able to settle the case $k = 2$, we show a polynomial-time solution under the additional constraint that only substrings of length *exactly* equal to 2 are captured by the attractor set. We denote with the name *k-sharp attractor* this variant. Formally, we define a *k-sharp* attractor of a string $S \in \Sigma^n$ to be a set of positions $\Gamma \subseteq [1..n]$ such that every substring $S[i..j]$ with $j-i+1 = k$ has an occurrence $S[i'..j'] = S[i..j]$ with $j'' \in [i'..j']$ for some $j'' \in \Gamma$. In other words, a k-sharp-attractor is a subset that covers all substrings of length exactly k. By Minimum-k-Sharp-Attractor we denote the optimization problem of finding the smallest k-sharp attractor of a given input string, and by k-Sharp-Attractor $= \{\langle T, p\rangle \mid \text{String } T \text{ has a } k\text{-sharp-attractor of size} \leq p\}$ we denote the corresponding decision problem. In the full version [15] of this paper we show that, for any constant $k \geq 3$, k-Sharp-Attractor is NP-complete. The proof is an adaptation of the original one proposed for k-attractors in [16], and is based on a reduction from set cover. Here we show that for $k = 2$ the problem can be solved in polynomial time (again, the case $k = 1$ is trivially solvable in linear time).

▶ **Theorem 25.** *Minimum 2-sharp-attractor is in P.*

Proof. It is easy to show that 2-sharp-attractor is in P by a reduction to edge cover. Given a string S, let $V \subseteq \Sigma^2$ be the set of strings of length 2 that occur at least once in S. For every substring of length 3 of the form xyz, add the edge (xy, yz) to the edge-set E, and add self-loops for the first and last pair.

A position $\gamma \in \Gamma$ thus corresponds to an edge, e_γ, and it is easy to see that Γ is a 2-sharp-attractor if and only if $\{e_\gamma | \gamma \in \Gamma\}$ is an edge cover.

The number of vertices and edges in this graph are both $\leq n$, so a minimum edge cover can be found in $\mathcal{O}(n\sqrt{n})$ time [19]. ◀

References

1 Djamal Belazzougui. Linear time construction of compressed text indices in compact space. In *Annual Symposium on Theory of Computing (STOC)*, pages 148–193. ACM, 2014.

2 Djamal Belazzougui, Fabio Cunial, Juha Kärkkäinen, and Veli Mäkinen. Linear-time string indexing and analysis in small space. *arXiv preprint 1609.06378*, 2016.

3 Anselm Blumer, Janet Blumer, David Haussler, Ross McConnell, and Andrzej Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *Journal of the ACM*, 34(3):578–595, 1987.

4 Michael Burrows and David J. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

5 David Clark. *Compact Pat trees.* PhD thesis, University of Waterloo, 1998.

6 Maxime Crochemore and Renaud Vérin. Direct construction of compact directed acyclic word graphs. In *Combinatorial Pattern Matching (CPM)*, pages 116–129. Springer, 1997.

7 Martin Farach. Optimal suffix tree construction with large alphabets. In *Annual Symposium on Foundations of Computer Science (FOCS)*, pages 137–143. IEEE, 1997.

8 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *Journal of the ACM*, 52(4):552–581, 2005.

9 Johannes Fischer and Volker Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM Journal on Computing*, 40(2):465–492, 2011.

10 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Annual Symposium on Discrete Algorithms (SODA)*, pages 841–850. SIAM, 2003.

11 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35(2):378–407, 2005.

12 Prosenjit Gupta, Ravi Janardan, and Michiel Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. *Journal of Algorithms*, 19(2):282–317, 1995.

13 Guy Joseph Jacobson. *Succinct static data structures*. PhD thesis, Carnegie Mellon University, 1988.

14 Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *Journal of the ACM*, 53(6):918–936, 2006.

15 Dominik Kempa, Alberto Policriti, Nicola Prezza, and Eva Rotenberg. String attractors: Verification and optimization. *arXiv*, 2018. `arXiv:1803.01695`.

16 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: String attractors. In *Annual Symposium on Theory of Computing (STOC)*, pages 827–840. ACM, 2018.

17 John C. Kieffer and En-Hui Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3):737–754, 2000.

18 Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

19 Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *Annual Symposium on Foundations of Computer Science (SFCS)*, pages 17–27. IEEE, 1980.

20 J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Space-efficient construction of compressed indexes in deterministic linear time. In *Annual Symposium on Discrete Algorithms (SODA)*, pages 408–424. SIAM, 2017.

21 Gonzalo Navarro. Wavelet trees for all. *Journal of Discrete Algorithms*, 25:2–20, 2014.

22 Gonzalo Navarro. *Compact data structures: A practical approach*. Cambridge University Press, 2016.

23 James A. Storer and Thomas G. Szymanski. Data compression via textual substitution. *Journal of the ACM*, 29(4):928–951, 1982.

24 Peter Weiner. Linear pattern matching algorithms. In *Annual Symposium on Switching and Automata Theory (SWAT/FOCS)*, pages 1–11. IEEE, 1973.

Data Reduction for Maximum Matching on Real-World Graphs: Theory and Experiments

Viatcheslav Korenwein
TU Berlin, Institut für Softwaretechnik und Theoretische Informatik, Berlin, Germany

André Nichterlein
TU Berlin, Institut für Softwaretechnik und Theoretische Informatik, Berlin, Germany
andre.nichterlein@tu-berlin.de

Rolf Niedermeier
TU Berlin, Institut für Softwaretechnik und Theoretische Informatik, Berlin, Germany
rolf.niedermeier@tu-berlin.de

Philipp Zschoche
TU Berlin, Institut für Softwaretechnik und Theoretische Informatik, Berlin, Germany
zschoche@tu-berlin.de

Abstract

Finding a maximum-cardinality or maximum-weight matching in (edge-weighted) undirected graphs is among the most prominent problems of algorithmic graph theory. For n-vertex and m-edge graphs, the best known algorithms run in $\widetilde{O}(m\sqrt{n})$ time. We build on recent theoretical work focusing on linear-time data reduction rules for finding maximum-cardinality matchings and complement the theoretical results by presenting and analyzing (thereby employing the kernelization methodology of parameterized complexity analysis) linear-time data reduction rules for the positive-integer-weighted case. Moreover, we experimentally demonstrate that these data reduction rules provide significant speedups of the state-of-the art implementation for computing matchings in real-world graphs: the average speedup is 3800% in the unweighted case and "just" 30% in the weighted case.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Maximum-cardinality matching, maximum-weight matching, linear-time algorithms, preprocessing, kernelization, parameterized complexity analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.53

Related Version `http://arxiv.org/abs/1806.09683`

Acknowledgements We are grateful to anonymous reviewers of *ESA '18* for constructive and detailed feedback.

1 Introduction

In their book chapter on weighted matching, Korte and Vygen [11] write that "weighted matching appears to be one of the 'hardest' combinatorial optimization problems that can be solved in polynomial time". Correspondingly, the design and analysis of matching algorithms plays a pivotal role in algorithm theory as well as in practical computing. Complementing the rich literature on matching algorithms (see Coudert, Ducoffe, and Popa [6] and Duan, Pettie, and Su [8] for recent accounts, the latter also providing a literature overview), in this work we focus on efficient linear-time data reduction rules that may help to speed up

© Viatcheslav Korenwein, André Nichterlein, Rolf Niedermeier, and Philipp Zschoche;
licensed under Creative Commons License CC-BY

superlinear matching algorithms. Notably, while recent breakthrough results on matching (including linear-time approximation algorithms [7]) focus on the theory side, we study both theory and practice, thereby achieving gains on both sides.

To achieve our results, we follow and significantly extend recent purely theoretical work [13] presenting and analyzing linear-time data reductions for the unweighted case. More specifically, on the theoretical side we complement these results by performing an analysis for the weighted case (turning out to become more technical); on the practical side, we demonstrate that these data reduction rules may serve to speed up the state-of-the-art matching solver due to Kolmogorov [10]. Similar data reduction rules have been implemented for finding maximum independent sets and minimum vertex covers [2, 5], leading to significant speedups of algorithms for both problems.

Formally, we study the following two problems; note that we formulate them as decision problems since this better fits with presenting our theoretical part where we prove kernelization results (thereby employing the framework of parameterized complexity analysis); our data reduction rules also work and are implemented for the optimization versions.

Maximum-Cardinality Matching
Input: An undirected graph $G = (V, E)$ and $s \in \mathbb{N}$.
Question: Is there a size-s subset $M \subseteq E$ of nonoverlapping (that is, pairwise vertex-disjoint) edges?

Maximum-Weight Matching
Input: An undirected graph $G = (V, E)$, edge weights $\omega\colon E \to \mathbb{N}$, and $s \in \mathbb{N}$.
Question: Is there a subset $M \subseteq E$ of nonoverlapping edges of weight $\sum_{e \in M} \omega(e) \geq s$?

We remark that all our results extend to the case of rational weights; however, integers are easier to cope with and are used in the implementation of Kolmogorov [10].

Our contributions. We lift known kernelization results [13] for Maximum-Cardinality Matching to Maximum-Weight Matching. To this end, we provide algorithms to efficiently apply our newly developed data reduction rules. Herein, we have a particular eye on exhaustively applying the data reduction rules in linear time, which seems imperative in an effort to practically improve matching algorithms. Hence, our main theoretical contribution lies in developing efficient algorithms implementing the data reduction rules, thereby also showing a purely theoretical guarantee on the amount of data reduction that can be achieved in the worst case (this is also known as kernelization in parameterized algorithmics)[1]. We proceed by implementing and testing the data reduction algorithms for Maximum-Cardinality Matching and Maximum-Weight Matching, thereby demonstrating their significant practical effectiveness. More specifically, combining them in form of preprocessing with Kolmogorov's state-of-the-art solver [10, 16] yield partially tremendous speedups on sparse real-world graphs (taken from the SNP library [12]). Concretely, comparing Kolmogorov's algorithm with and without our data reduction algorithms, the average speedup is 3800% in the unweighted case and "only" 30% in the weighted case.

Notation. We use standard notation from graph theory. All graphs considered in this work are simple and undirected. For a graph G, we denote with $E(G) = E$ the edge set.

[1] We clearly remark, however, that our theoretical findings (that is, kernel size upper bounds based on the feedback edge set number) are too weak in order to fully explain the practical success of the data reduction rules.

We write uv to denote the edge $\{u, v\}$ and $G - v$ to denote the graph obtained from G by removing v and all its incident edges. A feedback edge set of a graph G is a set X of edges such that $G - X$ is a tree or forest. The feedback edge number denotes the size of a minimum feedback edge set. A *matching* in a graph is a set of pairwise disjoint edges. Let G be a graph and let $M \subseteq E(G)$ be a matching in G. We denote by $\mathrm{match}(G)$ the maximum-cardinality matching respectively the maximum-weight matching in G, depending on whether we have edge weights or not. If there are edge weights $\omega \colon E \to \mathbb{N}$, then for a matching M we denote by $\omega(M) = \sum_{e \in M} \omega(e)$ the weight of M. Moreover, we denote with $\omega(G)$ the weight of a maximum-weight matching $\mathrm{match}(G)$, i. e. $\omega(G) = \omega(\mathrm{match}(G))$. A vertex $v \in V$ is called *matched* with respect to M if there is an edge in M containing v, otherwise v is called *free* with respect to M. If the matching M is clear from the context, then we omit "with respect to M".

Kernelization. A *parameterized problem* is a set of instances (I, k) where $I \in \Sigma^*$ for a finite alphabet Σ, and $k \in \mathbb{N}$ is the *parameter*. We say that two instances (I, k) and (I', k') of parameterized problems P and P' are *equivalent* if (I, k) is a yes-instance for P if and only if (I', k') is a yes-instance for P'. A *kernelization* is an algorithm that, given an instance (I, k) of a parameterized problem P, computes in polynomial time an equivalent instance (I', k') of P (the *kernel*) such that $|I'| + k' \leq f(k)$ for some computable function f. We say that f measures the *size* of the kernel, and if $f(k) \in k^{O(1)}$, then we say that P admits a polynomial kernel. Often, a kernel is achieved by applying polynomial-time executable data reduction rules. We call a data reduction rule $\mathcal{R}$ *correct* if the new instance (I', k') that results from applying $\mathcal{R}$ to (I, k) is equivalent to (I, k). An instance is called *reduced* with respect to some data reduction rule if further application of this rule has no effect on the instance.

2 Kernelization Algorithms

In this section, we recall the data reduction rules for Maximum-Cardinality Matching and show how to lift them to Maximum-Weight Matching. For Maximum-Cardinality Matching two simple data reduction rules are due to a classic result of Karp and Sipser [9]. They deal with vertices of degree at most two.

▶ **Reduction Rule 2.1** ([9]). *Let $v \in V$. If $\deg(v) = 0$, then delete v. If $\deg(v) = 1$, then delete v and its neighbor, and decrease the solution size s by one.*

▶ **Reduction Rule 2.2** ([9]). *Let v be a vertex of degree two and let u, w be its neighbors. Then remove v, merge u and w, and decrease the solution size s by one.*

In each application of the two data reduction rules the considered vertex v is matched (hence the decrease of the solution size). When applying Reduction Rule 2.1, then v is matched to its only neighbor u. For Reduction Rule 2.2 the situation is not so clear as v is matched to u or to w depending on how the maximum-cardinality matching in the rest of the graph looks like. Thus, one can only fix the matching edge with endpoint v (in the original graph) in a postprocessing step.

Both of the above data reduction rules can be exhaustively applied in linear time. While for Reduction Rule 2.1 this is easy to see, for Reduction Rule 2.2 the algorithm needs further ideas [3]. Using the above data reduction rules, one can show a kernel with respect to the parameter feedback edge number, that is, the size of a minimum feedback edge set.

▶ **Theorem 1** ([13]). Maximum-Cardinality Matching *admits a linear-time computable linear-size kernel with respect to the parameter feedback edge number k.*

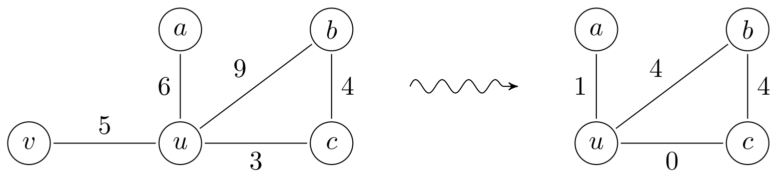


Figure 1 Left: Input graph. Right: The graph after applying Reduction Rule 2.4 to vertex v.

Applying the $O(m\sqrt{n})$-time algorithm for Maximum-Cardinality Matching [14] altogether yields an $O(n+m+k^{1.5})$-time algorithm, where k is the feedback edge number.

Weighted Matching. In the remainder of this section, we show how to lift Theorem 1 to the weighted case. Reduction Rules 2.1 and 2.2 are based on the simple observation that for every vertex $v \in V$ of degree at least one, there exists a maximum-cardinality matching containing v: If v is not matched, then take an arbitrary neighbor u of v, remove the edge containing u from a maximum-cardinality matching, and add the edge uv. This observation does not hold in the weighted case – see e. g. Figure 1 (left side) where the only maximum-weight matching $\{au, bc\}$ leaves v free. Thus, we need new ideas to obtain data reduction rules for the weighted case.

Vertices of degree at most one. We start with the simple case of dealing with vertices of degree at most one. Here, the following data reduction rule is obvious.

▶ **Reduction Rule 2.3.** *If* $\deg(v) = 0$ *for a vertex* $v \in V$*, then delete* v*. If* $\omega(e) = 0$ *for an edge* $e \in E$*, then delete* e*.*

Next, we show how to deal with degree-one vertices, see Figure 1 for a visualization.

▶ **Reduction Rule 2.4.** *Let* $G = (V, E)$ *be a graph with non-negative edge weights* $\omega \colon E \to \mathbb{N}$*. Let* v *be a degree-one vertex and let* u *be its neighbor. Then delete* v*, set the weight of every edge* e *incident with* u *to* $\max\{0, \omega(e) - \omega(uv)\}$*, and decrease the solution value* s *by* $\omega(uv)$*.*

While proving the correctness of this rule (see next lemma) is relatively straightforward, the naive algorithm to exhaustively apply Reduction Rule 2.4 is too slow for our purpose: If the edge weights are adjusted immediately after deleting v, then exhaustively applying the rule to a star requires $\Theta(n^2)$ time. However, as we subsequently show, Reduction Rule 2.4 can be exhaustively applied in linear time.

▶ **Lemma 2.** *Reduction Rule 2.4 is correct.*

Due to space restrictions, we defer the poof of Lemma 2 to the arXiv version.

▶ **Lemma 3.** *Reduction Rule 2.4 can be exhaustively applied in* $O(n + m)$ *time.*

Proof. The basic idea of the algorithm exhaustively applying Reduction Rule 2.4 in linear time is as follows: We store in each vertex a number indicating the weight of the heaviest incident edge removed due to Reduction Rule 2.4. Then, whenever we want to access the "current" weight of an edge e, then we subtract from $\omega(e)$ the two numbers stored in the two incident vertices. Once Reduction Rule 2.4 is no more applicable, then we update the edge weights to get rid of the numbers in the vertices in order to create a Maximum-Weight Matching instance.

The details of the algorithm are as follows. First, in $O(n+m)$ time we collect all degree-one vertices in a list L and initialize for each vertex v a counter $c(v) := 0$. Then, we process L one by one. For a degree-one vertex $v \in L$, let u be its neighbor. We decrease s by $\max\{0, w(uv) - c(u) - c(v)\}$, then set $c(u) := c(u) + \max\{0, w(uv) - c(u) - c(v)\}$, and then delete v. If after the deletion of v its neighbor u has degree one, then u is added to L. Thus, after at most n steps, each one doable in constant time, we processed L. When L is empty, then in $O(m)$ time we update for each edge uv its weight $w(uv) := \max\{0, w(uv) - c(u) - c(v)\}$. This finishes the description of the algorithm.

Observe that we have the following invariant when processing the list L: the weight of an edge uv is $\max\{0, w(uv) - c(u) - c(v)\}$. With this invariant, it is easy to see that the algorithm indeed applies Reduction Rule 2.4 exhaustively. ◀

Note that after applying Reduction Rule 2.4 we can have weight-zero edges and thus Reduction Rule 2.3 might become applicable. We do not know whether Reduction Rules 2.3 and 2.4 *together* can be applied exhaustively in linear time. However, for the kernel we present in the end of this section it is sufficient to apply Reduction Rule 2.4 exhaustively.

Vertices of degree two. Lifting Reduction Rule 2.2 to the weighted case is more delicate than lifting Reduction Rule 2.1 to Reduction Rules 2.3 and 2.4. The reason is that the two incident edges might have different weights. As a consequence, we cannot decide locally what to do with a degree-two vertex. Instead, we process multiple degree-two vertices at once. To this end, we use the following notation.

▶ **Definition 4.** Let G be a graph. A path $P = v_0 v_1 \dots v_\ell$ is a *maximal path* in G if $\ell \geq 3$, the inner vertices $v_1, v_2, \dots, v_{\ell-1}$ all have degree two in G, but the endpoints v_0 and v_ℓ do not, that is, $\deg_G(v_1) = \dots = \deg_G(v_{\ell-1}) = 2$, $\deg_G(v_0) \neq 2$, and $\deg_G(v_\ell) \neq 2$.

▶ **Definition 5.** Let G be a graph. A cycle $C = v_0 v_1 \dots v_\ell v_0$ is a *pending cycle* in G if at most one vertex in C does *not* have degree two in G.

The reason to study maximal paths and pending cycles is that we can compute a maximum-weight matching in these graphs quickly, as stated next. This allows us to preprocess all vertices in a maximal path or a pending cycle at once.

▶ **Observation 6.** Maximum-Weight Matching *can be solved in $O(n)$ time on paths and cycles.*

Proof. If the input graph G is a path, then by exhaustively applying Reduction Rules 2.3 and 2.4, we can compute a maximum-weight matching. Otherwise, if G is a cycle, then we take an arbitrary edge e and distinguish two cases. First, we take e into a matching and remove both endpoints from the graph. In the resulting path, we compute in linear time a maximum-weight matching M. Second, we delete e and obtain a path for which we compute in linear time a maximum-weight matching M'. We then simply choose between $M \cup \{e\}$ and M' the heavier matching as the result. ◀

Now, using Observation 6, we introduce data reduction rules for maximal paths and pending cycles. Both rules are based on a similar idea which is easier to explain for a pending cycle. Let C be a pending cycle and $u \in C$ the degree-at-least-three vertex in C. Then there are two cases: u is matched with a vertex not in C or it is not. Now let M be a maximum-weight matching for G, and let M' be a maximum-weight matching with the constraint that u is matched to a vertex outside C. Clearly, $M \cap E(C)$ is at least as

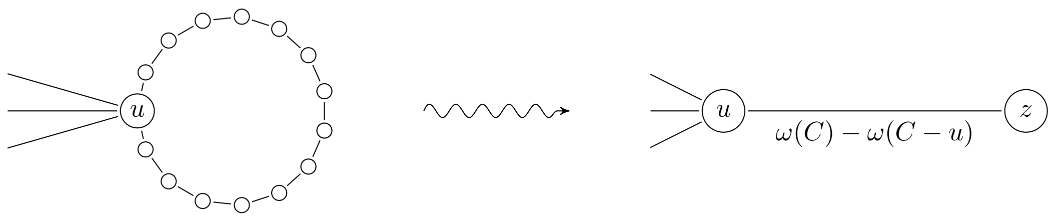


Figure 2 Left: A pending cycle C with u being the vertex of degree more than three. Right: The graph after applying Reduction Rule 2.5 where s is reduced by $\omega(C - u)$.

large as $M' \cap E(C)$. Looking only at C, all that we need to know is the difference of the weights of these two matchings. This can be encoded with one vertex z which replaces the whole cycle C (see Figure 2 for an illustration). Then, matching z corresponds to taking the matching in C and not matching z corresponds to taking the matching in $C - u$. Formalizing this idea, we arrive at the following data reduction rule.

▶ **Reduction Rule 2.5.** *Let G be a graph with non-negative edge weights. Let C be a pending cycle in G, where $u \in C$ has degree at least three in G. Then replace C by an edge uz with $\omega(uz) = \omega(C) - \omega(C - u)$ and decrease the solution value s by $\omega(C - u)$.*

▶ **Lemma 7.** *Reduction Rule 2.5 is correct.*

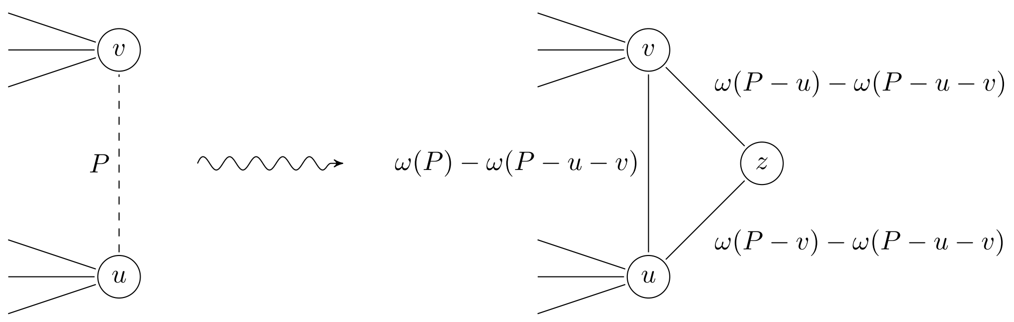
Proof. Let C be a pending cycle in G where $u \in C$ has degree at least three in G and let G' be the graph obtained applying Reduction Rule 2.5 to C. We show $\omega(G') = \omega(G) - \omega(C - u)$. Let M be a maximum-weight matching in G. Let $M_{\overline{C}} := M \setminus E(C)$. Observe that $\omega(M_{\overline{C}}) = \omega(M) - \omega(M \cap E(C)) \geq \omega(G) - \omega(C)$. If u is matched with respect to $M_{\overline{C}}$, then we have $M_{\overline{C}} = M \setminus E(C - u)$. Hence, $\omega(G') \geq \omega(M_{\overline{C}}) \geq \omega(G) - \omega(C - u)$. If u is free with respect to $M_{\overline{C}}$, then $M_{\overline{C}} \cup \{uz\}$ is a matching in G' with weight at least $(\omega(G) - \omega(C)) + (\omega(C) - \omega(C - u)) = \omega(G) - \omega(C - u)$. Hence, in both cases we have $\omega(G') \geq \omega(G) - \omega(C - u)$.

Conversely, let M' be a maximum-weight matching in G'. Recall that, for an edge-weighted graph H, $\operatorname{match}(H)$ denotes a maximum-weight matching in H. If $uz \in M'$, then $(M' \setminus \{uz\}) \cup \operatorname{match}(C)$ is a matching in G with $\omega(G') - (\omega(C) - \omega(C - u)) + \omega(C) = \omega(G') + \omega(C - u)$. Hence, $\omega(G) \geq \omega(G') + \omega(C - u)$. If $uz \notin M'$, then $M' \cup \operatorname{match}(C - u)$ is a matching in G with weight at least $\omega(G') + \omega(C - u)$. Again, in both cases we have $\omega(G) \geq \omega(G') + \omega(C - u)$. Combined with $\omega(G') \geq \omega(G) - \omega(C - u)$, we arrive at $\omega(G') = \omega(G) - \omega(C - u)$. ◀

The basic idea for maximal paths is the same as for pending cycles. The difference is that we have to distinguish four cases depending on whether or not the two endpoints u and v of a maximal path P are matched within P. To simplify the notation, we set $\omega(uv) = 0$ if the edge uv does not exist in G. Furthermore, $P - u - v$ denotes the path obtained from removing in P the vertices u and v. This avoids some trivial case distinctions.

Figure 3 visualizes the next data reduction rule.

▶ **Reduction Rule 2.6.** *Let $G = (V, E)$ be a graph with non-negative edge weights $\omega \colon E \to \mathbb{N}$. Let P be a maximal path in G with endpoints u and v. Then remove all vertices in P except u and v, add a new vertex z and, if not already existing, add the edge uv. Furthermore, set $\omega(uz) := \omega(P - v) - \omega(P - u - v)$, $\omega(vz) := \omega(P - u) - \omega(P - u - v)$, and $\omega(uv) := \max\{\omega(uv), \omega(P) - \omega(P - u - v)\}$, and decrease the solution value s by $\omega(P - u - v)$.*



v
P
u
v
$\omega(P-u)-\omega(P-u-v)$
$\omega(P)-\omega(P-u-v)$
z
$\omega(P-v)-\omega(P-u-v)$
u

Figure 3 Applying Reduction Rule 2.6 on a path P with endpoints u and v (where u and v are not adjacent). The four choices for u and v on whether or not they are matched to a vertex within the path are reflected by the three (full) edges on the right where at most one can be taken into a matching. Since the edge uv is not contained in the input graph the weight of the edge uv in the reduced graph simplifies to the displayed value.

▶ **Lemma 8.** *Reduction Rule 2.6 is correct.*

Proof. Let G be the input graph with a maximal path P with endpoints u and v. Furthermore, let G' be the reduced instance with z defined as in the data reduction rule. We show that $\omega(G') = \omega(G) - \omega(P-u-v)$.

Let M be a maximum-weight matching for G. We define $M_{\overline{P}} := M \setminus E(P)$. Observe that $\omega(M_{\overline{P}}) = \omega(M) - \omega(M \cap E(P)) \geq \omega(G) - \omega(P)$ We distinguish four cases.

1. If both u and v are matched with respect to $M_{\overline{P}}$, then $M_{\overline{P}} = M \setminus E(P-u-v)$ and hence $\omega(M_{\overline{P}}) = \omega(M) - \omega(M \cap E(P-u-v)) \geq \omega(G) - \omega(P-u-v)$.
2. If u is matched and v is free with respect to $M_{\overline{P}}$, then $M_{\overline{P}} = M \setminus E(P-u)$ and hence $\omega(M_{\overline{P}}) \geq \omega(G) - \omega(P-u)$. Thus, $M_{\overline{P}} \cup \{vz\}$ is a matching of weight at least $(\omega(G) - \omega(P-u)) + (\omega(P-u) - \omega(P-u-v)) = \omega(G) - \omega(P-u-v)$.
3. If v is matched and u is free with respect to $M_{\overline{P}}$, then $M_{\overline{P}} = M \setminus E(P-v)$ and hence $\omega(M_{\overline{P}}) \geq \omega(G) - \omega(P-v)$. Thus, $M_{\overline{P}} \cup \{uz\}$ is a matching of weight at least $(\omega(G) - \omega(P-v)) + (\omega(P-v) - \omega(P-u-v)) = \omega(G) - \omega(P-u-v)$.
4. Finally, if both u and v are free with respect to $M_{\overline{P}}$, then $M_{\overline{P}} \cup \{uv\}$ is a matching of weight at least $(\omega(G) - \omega(P)) + (\omega(P) - \omega(P-u-v)) = \omega(G) - \omega(P-u-v)$.

Thus in each case we have $\omega(G') \geq \omega(G) - \omega(P-u-v)$.

Conversely, let M' be a maximum-weight matching for G'. We define $\overline{M'} := M' \setminus \{uz, vz, uv\}$. Again, we distinguish four cases.

1. If both u and v are matched with respect to $\overline{M'}$, then $\overline{M'} = M'$. Hence, $\overline{M'} \cup \text{match}(P-u-v)$ is a matching in G with weight at least $\omega(G') + \omega(P-u-v)$.
2. If u is matched and v is free with respect to $\overline{M'}$, then w.l.o.g. $vz \in M'$. Hence, $\overline{M'} \cup \text{match}(P-u)$ is a matching in G with weight at least $\omega(G') - (\omega(P-u) - \omega(P-u-v)) + \omega(P-u) = \omega(G') + \omega(P-u-v)$.
3. If u is matched and v is free with respect to $\overline{M'}$, then w.l.o.g. $uz \in M'$. Hence, $\overline{M'} \cup \text{match}(P-v)$ is a matching in G with weight at least $\omega(G') - (\omega(P-v) - \omega(P-u-v)) + \omega(P-v) = \omega(G') + \omega(P-u-v)$.
4. Finally, if both u and v are free with respect to $\overline{M'}$, then w.l.o.g $uv \in M'$ as $\omega(uv) \geq \omega(uz)$ and $\omega(uv) \geq \omega(vz)$. Now, we encounter two subcases.
   a. If $\omega(uv) > \omega(P) - \omega(P-u-v)$, then the edge uv is in G and in G', having the same weight in both graphs. Then, $M' \cup \text{match}(P-u-v)$ is a matching in G with weight at least $\omega(G') + \omega(P-u-v)$.
   b. Otherwise, $\overline{M'} \cup \text{match}(P)$ is a matching in G with weight at least $\omega(G') - (\omega(P) - \omega(P-u-v)) + \omega(P) = \omega(G') + \omega(P-u-v)$.

Hence, in all cases we have $\omega(G) \geq \omega(G') + \omega(P - u - v)$. Combined with $\omega(G') \geq \omega(G) + \omega(P - u - v)$, we can infer that $\omega(G') = \omega(G) - \omega(P - u - v)$. ◀

▶ **Lemma 9.** *Reduction Rules 2.5 and 2.6 can be exhaustively applied in $O(n + m)$ time.*

Proof. First, we collect in $O(n+m)$ time all maximal paths and all pending cycles [4, Lemma 2]. Given a maximal path or a pending cycle on ℓ vertices due to Observation 6 one can compute the necessary maximum-weight matchings (at most four) in $O(\ell)$ time. Moreover, replacing the maximal path or the pending cycle by the respective structure is doable in $O(\ell)$ time. Applying Reduction Rules 2.5 and 2.6 does not create new maximal paths (recall that a maximal path needs at least two vertices of degree two) or pending cycles. Thus, as all maximal paths and pending cycles combined contain at most n vertices, Reduction Rules 2.5 and 2.6 can be exhaustively applied in $O(n + m)$ time. ◀

Each of Reduction Rules 2.3, 2.4, and 2.6 can be exhaustively applied in linear time; however, we do not know whether all these data reduction rules together can be exhaustively applied in linear time. Note that after applying Reduction Rule 2.5 Reduction Rule 2.4 might become applicable. For our problem kernel below, however, Lemmas 3 and 9 are sufficient. In contrast to this subsequent theoretical result, in our experimental part it proved beneficial to apply the rules exhaustively in order to remove as many vertices and edges as possible.

▶ **Theorem 10.** Maximum-Weight Matching *admits a linear-time computable $20k$-vertex and $22k$-edge kernel with respect to the parameter feedback edge number k.*

Proof. The kernelization algorithm works as follows: First, exhaustively apply Reduction Rule 2.3 in $O(n + m)$ time. Second, exhaustively apply Reduction Rules 2.5 and 2.6 in $O(n + m)$ time (see Lemma 9). Third, exhaustively apply Reduction Rule 2.4 in $O(n + m)$ time (see Lemma 3). Without loss of generality, one can assume that the input graph does not contain a cycle where each vertex has degree two, because otherwise this cycle can be solved independently in linear time (see Observation 6). Note that when applying the rules in this order, the resulting graph $G = (V, E)$ does not contain any degree one-vertices, maximal paths, or pending cycles.

We claim that G has less than $20k$ vertices and $22k$ edges. First, note that the input graph contains at most k maximal paths [4, Lemma 1]. Thus, a feedback edge set $X \subseteq E$ for G contains at most $2k$ edges (each application of Reduction Rule 2.6 increases the feedback edge set by one). Denote with V^1_{G-X}, V^2_{G-X}, and $V^{\geq 3}_{G-X}$ the vertices that have degree one, two, and more than two in $G - X$. Observe that all vertices in the reduced graph G have degree at least two since it is reduced with respect to Reduction Rules 2.3 and 2.4. Thus, $|V^1_{G-X}| \leq 4k$ as each leaf in $G - X$ has to be incident to an edge in X. Next, since $G - X$ is a forest (or tree), we have $|V^{\geq 3}_{G-X}| < |V^1_{G-X}|$ and thus $|V^{\geq 3}_{G-X}| < 4k$. Finally, each degree-two vertex in G needs two neighbors of degree at least three since G is reduced with respect to Reduction Rules 2.5 and 2.6. Thus, the vertices in V^2_{G-X} are either incident to an edge in X or adjacent to one of the at most $|V^{\geq 3}_{G-X}| + 4k$ vertices in G that have degree at least three. The sum over all degrees of vertices in $V^{\geq 3}_{G-X}$ is

$$\begin{aligned}\sum_{v \in V^{\geq 3}_{G-X}} \deg_{G-X}(v) = 2m - \sum_{v \in V^{=2}_{G-X} \cup V^{=1}_{G-X}} \deg_{G-X}(v) &\leq 2(n-1) - 2|V^{=2}_{G-X}| - |V^{=1}_{G-X}| \\ &= 2|V^{\geq 3}_{G-X}| + |V^1_{G-X}| - 2 < 12k.\end{aligned}$$

It follows that $|V^2_{G-X}| \leq 16k$. Thus, the number of vertices in G is $|V^1_{G-X}| + |V^2_{G-X}| + |V^{\geq 3}_{G-X}| \leq 20k$. Since $G - X$ is a forest, it follows that G has at most $|V| + 2k \leq 22k$ edges. ◀

Table 1 A selection of our test graphs from SNAP [12] with their respective size.

Graph	n	edges	Graph	n	edges
email-Eu-core	1,005	16,064	p2p-Gnutella08	6,301	20,777
p2p-Gnutella25	22,687	54,705	ca-CondMat	23,133	93,439
soc-...dot090221	82,141	$3.49 \cdot 10^5$	soc-Slashdot0811	77,360	$4.69 \cdot 10^5$
com-dblp	$3.2 \cdot 10^5$	$1.05 \cdot 10^6$	twitter-combined	81,306	$1.34 \cdot 10^6$
amazon0505	$4.1 \cdot 10^5$	$2.44 \cdot 10^6$	roadNet-CA	$2 \cdot 10^6$	$2.77 \cdot 10^6$
wiki-topcats	$1.8 \cdot 10^6$	$2.54 \cdot 10^7$	soc-LiveJournal1	$4.8 \cdot 10^6$	$4.29 \cdot 10^7$

3 Experimental Evaluation

In this section we provide an experimental evaluation of the data reduction rules on real-world graphs ranging from a few thousand vertices and edges to a hundred million vertices and edges. We analyze the effectiveness and efficiency of the kernelization as well as the effect on the subsequently used state-of-the-art solver "blossom5" of Kolmogorov [10].

3.1 Setup & Implementation Details

Setup. Our program is written in C++14 and source code in available from `http://fpt.akt.tu-berlin.de/software/`. One can replicate all experiments by following the manual next to the source code. We ran all our experiments on an Intel(R) Xeon(R) CPU E5-1620 3.60 GHz machine with 64 GB main memory under the Debian GNU/Linux 7.0 operating system, where we compiled the program with GCC 5.4.0. All tested graphs are from the established SNAP [12] data set. See Table 1 for a sample list of graphs with their respective number of vertices and edges. The weighted graphs are generated from the unweighted graphs by adding edge-weights between 1 and 1000 chosen independently and uniformly at random. The full list is given in the arXiv version.

Implementation Details. The first step of our program is to read the graph into our data structure. On average this took 9% of the overall running time. When running blossom5 we also measured the time to handover the graph from our data structure to the solver's data structure, which took on average 4% of the overall running time.

We implemented kernelization algorithms for the unweighted and weighted case. The first kernelization is for Maximum-Cardinality Matching, which exhaustively applies Reduction Rules 2.1 and 2.2. Note that one can (theoretically) improve our implementation of Reduction Rule 2.2 by a linear-time algorithm of Bartha and Kresz [3]. However, our naive implementation proved to be sufficient. The second kernelization is for Maximum-Weight Matching. We use the algorithms described in Lemmas 3 and 9 to apply Reduction Rules 2.4 to 2.6. Deviating from the algorithm described in Theorem 10, based on empirical observations our program applies Reduction Rules 2.3 to 2.6 as long as possible. Hence, the kernelization does not run in linear time but further shrinks the input graph.

Note that all reported running times involving blossom5 are averages over 100 runs where we randomly permute vertex indices in the input. Although this permutation yields an isomorphic graph, we empirically observed that in the unweighted case the running time of blossom5 heavily depends on the permutation. For example, choosing a "good" or a "bad" permutation for the same graph yields speed ups of a factor 20 or more. (See the arXiv version for the results without this permutation.) When using blossom5 in the weighted case,

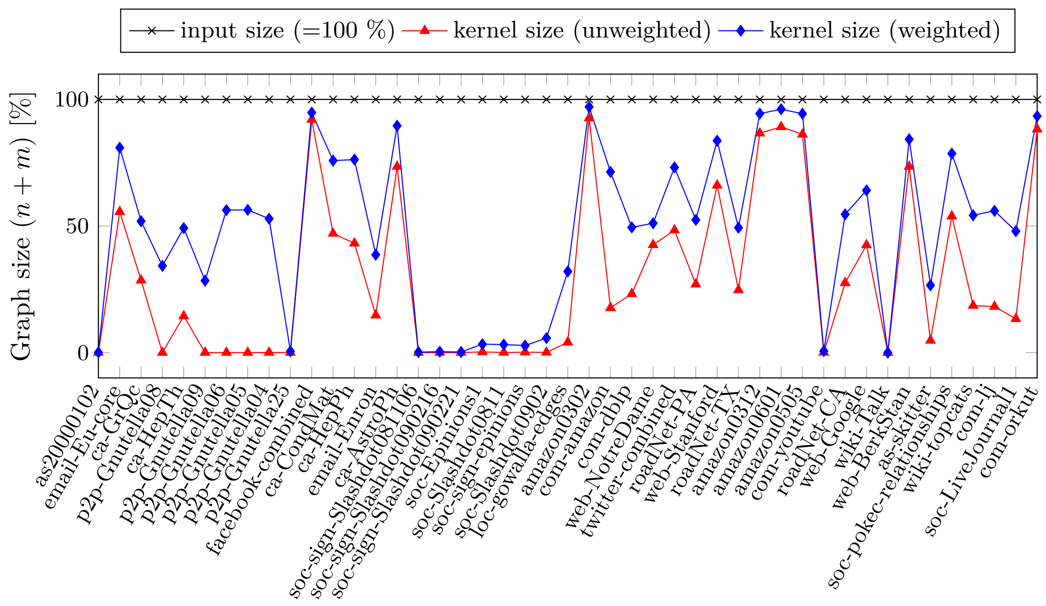


Figure 4 Percentage of the number of remaining edges and vertices after the respective kernelization algorithms (weighted, unweighted) relative to the numbers of vertices and edges in the input graph.

we did not observe this effect. For consistency, however, we take the average running time also in the weighted case. Note that our kernelization algorithm for the unweighted case was not at all affected by changing the permutation. In the weighted case, however, for different permutations the rules were applied in different order resulting in kernels slightly differing in size. The time for computing the random permutation is included in the times measured for reading and parsing of the graph.

3.2 Evaluation

We next present the results of our experimental evaluation starting with the size reduction and running time of the kernelization algorithm.

Kernel size. The effectiveness of our kernelization algorithms is displayed in Figure 4: Few graphs remain almost unchanged while other graphs are essentially solved by the kernelization algorithm. As expected, the kernelization algorithm for the unweighted case is much more effective than for the weighted case. On the 40 tested graphs, on average 70% of the vertices and edges are removed by the kernelization; the median is 81%. The least amenable graph was `amazon0302` with a size reduction of only 7%. In contrast, on 16 out of the 40 graphs the kernelization algorithm reduces more than 99% of the vertices and edges.

While the data reduction rules are less effective in the weighted case, they reduce the graphs on average still by 51% with the median value being a bit lower with 48%. The least amenable graph is again `amazon0302` with a size reduction of only 3%. Still, on seven out of the 40 graphs the kernelization algorithm reduces more than 99% of the vertices and edges.

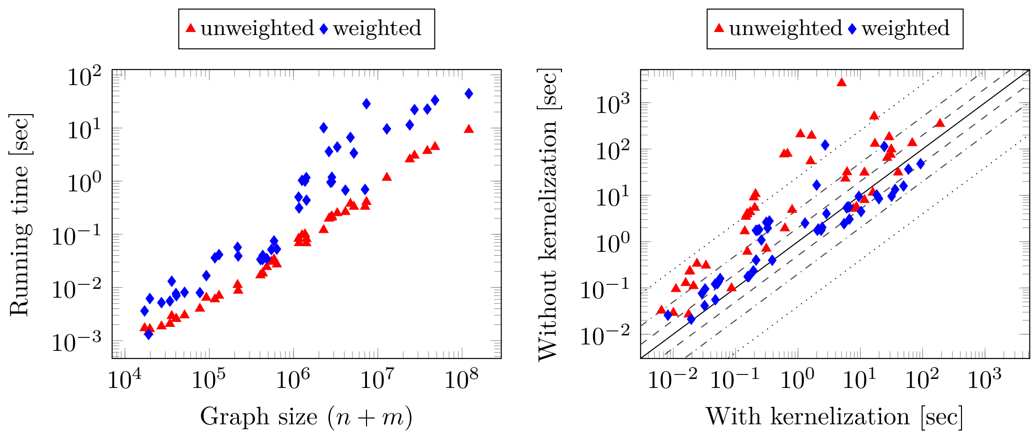


Figure 5 Left: Kernelization time depending on the graph sizes. Right: Running time comparison with and without using our kernelization algorithms before blossom5. The solid/dashed/dash dotted/dotted lines indicate a factor of 1/2/5/25 difference in the running time.

Kernelization time. We next discuss the running time of our kernelization algorithms on the test set. To this end, we consider the time spent on kernelization and the time spent on blossom5. We will, for now, omit the running times needed for reading and parsing the graph as these steps require on some instances more time than the kernelization algorithms.

As shown in Figure 5 (left), our kernelization algorithms are quite efficient. Even on the largest graphs with more than 120 million edges and vertices, the running time is less than 45 seconds in the weighted case and less than ten seconds in the unweighted case. In the unweighted case, our kernelization algorithm is always by at least a factor of 10 faster than blossom5 (on the first 40 graphs). Hence, applying the kernelization algorithm before the matching algorithm should – in the (unlikely) worst case of only few applications of the data reduction rules – only slightly increase the overall running time.

In the weighted case, however, our kernelization algorithm becomes slower than in the unweighted case. This is not surprising as the kernelization algorithm is more involved than the one for the unweighted case. Furthermore, blossom5 is significantly faster in the weighted case; on four graphs the matching algorithm is up to 2.5 times faster than our kernelization algorithm. However, on most graphs, our kernelization algorithm is still significantly faster than blossom5 (on average 17 times faster).

Running time comparison. We now compare the running time of only using blossom5 to first apply our kernelization algorithms and then use blossom5 on the kernel. Recall that all reported running times are averages over 100 runs. Since this 100 repetitions would have taken years for some graphs, we use only the 40 smallest graphs for the comparison. These graphs have between 12 thousand and 11 million edges.

Figure 5 (right) displays the results of the running time comparison. As one can clearly see, in the unweighted case the kernelization significantly accelerates the algorithm to find a maximum-cardinality matching. The speedup ranges from a factor 1.15 for the graph `facebook-combined` to a factor 525 for the graph `as-skitter` (the largest graph in this test set). The average speedup factor is 38, the median is 8.9.

For the weighted case, results are not as clear. With kernelization the algorithm is between 2.3 times slower for the graph `roadNet-CA` and 44.7 times faster for the graph `wiki-Talk`. However, the `wiki-Talk` is an exception as it is the only graph where the kernelization gave a speedup of a factor more than ten. The average speedup factor is 3.10 and the median is 1.3. On ten out of the 40 graphs the algorithm with kernelization was slower than without. As discussed above, there are even four graphs where blossom5 is even faster than our kernelization algorithm alone.

Summary. While the kernelization algorithms reduce the input graphs quite significantly in the weighted and unweighted case, the overall gain is very different in the two cases. When searching for a maximum-cardinality matching, we clearly recommend to always apply our kernelization algorithm. For the less clear weighted case, note that the kernelization is more frequently beneficial than it is not. However, the speedup is not as large as in the unweighted case and several instances are actually solved somewhat slower. There are several reasons for this behavior; some of which motivate future research and also lead to engineering challenges:

First, blossom5 is significantly faster on weighted graphs. We believe that the reason for this is in unweighted graphs there are a lot of symmetries, and unlucky tie-breaking seems to have a strong impact on blossom5. In the weighted case, the performance of blossom5 was much more consistent under permuting the vertices in the input graph. As a consequence, we believe that the following might speed up the algorithm: given an unweighted graph, introduce edge-weights such that a maximum-weight matching in the then weighted graph is also a maximum-cardinality matching in the unweighted graph. Using the famous Isolation Lemma [15] one might even enrich and support this with a theoretical analysis. As our focus was on data reduction, here we did not pursue this line of research yet.

Second, the kernelization algorithm in the weighted case is significantly slower than in the unweighted case (see Figure 5) as applying the rules is more involved. Note that Reduction Rules 2.1 and 2.2 only make changes in the local neighborhood of the affected vertices. This is not the case in the weighted case, where the application of Reduction Rules 2.4 to 2.6 involve iterations over all edges, see Lemmas 3 and 9. Hence, applying the data reduction rules exhaustively requires a larger overhead. Although some improvements in the implementation might be possible, an improved algorithmic approach to exhaustively apply the data reduction rules is needed. Is there a (quasi-)linear-time algorithm to exhaustively apply Reduction Rules 2.3 to 2.6?

4 Conclusion

Our work shows that it practically pays off to use (linear-time) data reduction rules for computing maximum matchings. The current state of the theoretical (kernel size upper bounds) analysis, however, is insufficient to fully explain this success. In future research, one might also study the combination of data reduction with linear-time approximation algorithms for matching [7]. Moreover, while our naive implementation for the unweighted case proved to be quite fast, we still work on improving the algorithm for the weighted case. In particular, parallelizing the kernelization algorithm seems promising for further speedups. We conclude with the following questions:

- Can exhaustive application of Reduction Rules 2.3 to 2.6 be realized in linear time?
- The “crown” data reduction rule known for the NP-hard VERTEX COVER problem [1] can be transferred to MAXIMUM-CARDINALITY MATCHING; however, our preliminary tests did not show a significant improvement of the kernelization algorithm. Is there any version of the “crown” data reduction rule with practical usefulness in matching computations?

References

1 Faisal N. Abu-Khzam, Michael R. Fellows, Michael A. Langston, and W. Henry Suters. Crown structures for vertex cover kernelization. *Theory of Computing Systems*, 41(3):411–430, 2007.

2 Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. *Theoretical Computer Science*, 609:211–225, 2016.

3 M. Bartha and M. Kresz. A depth-first algorithm to reduce graphs in linear time. In *Proceeding of the 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC' 09)*, pages 273–281. IEEE, 2009.

4 Matthias Bentert, Alexander Dittmann, Leon Kellerhals, André Nichterlein, and Rolf Niedermeier. Towards improving brandes' algorithm for betweenness centrality. *CoRR*, abs/1802.06701, 2018. URL: `http://arxiv.org/abs/1802.06701`.

5 Lijun Chang, Wei Li, and Wenjie Zhang. Computing A near-maximum independent set in linear time by reducing-peeling. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '17)*, pages 1181–1196. ACM, 2017.

6 David Coudert, Guillaume Ducoffe, and Alexandru Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '18)*, pages 2765–2784, 2018.

7 Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM*, 61(1):1:1–1:23, 2014.

8 Ran Duan, Seth Pettie, and Hsin-Hao Su. Scaling algorithms for weighted matching in general graphs. *ACM Transactions on Algorithms*, 14(1):8:1–8:35, 2018.

9 Richard M. Karp and Michael Sipser. Maximum matchings in sparse random graphs. In *Proceedings of the 22nd Annual IEEE Symposium on Foundations of Computer Science (FOCS '81)*, pages 364–375. IEEE, 1981.

10 Vladimir Kolmogorov. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1):43–67, 2009.

11 Bernd Korte and Jens Vygen. *Combinatorial Optimization – Theory and Algorithms*. Springer, 2018.

12 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, 2014.

13 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. The power of linear-time data reduction for maximum matching. In *Proceedings of the 42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, LIPIcs, pages 46:1–46:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

14 Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science (FOCS '80)*, pages 17–27. IEEE, 1980.

15 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

16 Sanne Wøhlk and Gilbert Laporte. Computational comparison of several greedy algorithms for the minimum cost perfect matching problem on large graphs. *Computers & OR*, 87:107–113, 2017.

Searching a Tree with Permanently Noisy Advice

Lucas Boczkowski
CNRS, IRIF, Univ. Paris Diderot, Paris, France
lucas.boczkowski@irif.fr

Amos Korman
CNRS, IRIF, Univ. Paris Diderot, Paris, France
amos.korman@irif.fr

Yoav Rodeh
Dep. of Physics of Complex Systems. Weizmann Institute, Rehovot, Israel
yoav.rodeh@gmail.com

Abstract

We consider a search problem on trees using unreliable guiding instructions. Specifically, an agent starts a search at the root of a tree aiming to find a treasure hidden at one of the nodes by an adversary. Each visited node holds information, called *advice*, regarding the most promising neighbor to continue the search. However, the memory holding this information may be unreliable. Modeling this scenario, we focus on a probabilistic setting. That is, the advice at a node is a pointer to one of its neighbors. With probability q each node is *faulty*, independently of other nodes, in which case its advice points at an arbitrary neighbor, chosen uniformly at random. Otherwise, the node is *sound* and points at the correct neighbor. Crucially, the advice is *permanent*, in the sense that querying a node several times would yield the same answer. We evaluate efficiency by two measures: The *move complexity* denotes the expected number of edge traversals, and the *query complexity* denotes the expected number of queries.

Let Δ denote the maximal degree. Roughly speaking, the main message of this paper is that a phase transition occurs when the *noise parameter* q is roughly $1/\sqrt{\Delta}$. More precisely, we prove that above the threshold, every search algorithm has query complexity (and move complexity) which is both exponential in the depth d of the treasure and polynomial in the number of nodes n. Conversely, below the threshold, there exists an algorithm with move complexity $\mathcal{O}(d\sqrt{\Delta})$, and an algorithm with query complexity $\mathcal{O}(\sqrt{\Delta}\log\Delta\log^2 n)$. Moreover, for the case of regular trees, we obtain an algorithm with query complexity $\mathcal{O}(\sqrt{\Delta}\log n\log\log n)$. For q that is below but close to the threshold, the bound for the move complexity is tight, and the bounds for the query complexity are not far from the lower bound of $\Omega(\sqrt{\Delta}\log_\Delta n)$.

In addition, we also consider a *semi-adversarial* variant, in which an adversary chooses the direction of advice at faulty nodes. For this variant, the threshold for efficient moving algorithms happens when the noise parameter is roughly $1/\Delta$. Above this threshold a simple protocol that follows each advice with a fixed probability already achieves optimal move complexity.

2012 ACM Subject Classification Theory of computation → Probabilistic computation, Theory of computation → Database theory, Theory of computation → Theory of randomized search heuristics

Keywords and phrases Data structures, Graph search, Average Case Analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.54

Related Version A full version of the paper is available at [5], `https://arxiv.org/abs/1611.01403`.

Funding This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 648032).

© Lucas Boczkowski, Amos Korman, and Yoav Rodeh;
licensed under Creative Commons License CC-BY

1 Introduction

This paper considers a basic search problem on trees, in which the goal is to find a treasure that is placed at one of the nodes by an adversary. Each node of the tree holds information, called *advice*, regarding which of its neighbors is closer to the treasure, and the search may consult the advice at some nodes in order to accelerate the search. Crucially, we assume that advice at nodes may be faulty with some probability. Many works consider noisy queries in the context of search, but it is typically assumed that queries can be resampled (see e.g., [12, 19, 4, 11]). In contrast, we assume that each location is associated with a single *permanent* advice. That is, faults are in the physical memory associated with the node, and hence querying the node again would yield the same answer. This difference is dramatic, as the search under our model does not allow for simple amplification procedures (similar to [7] albeit in the context of sorting). Searching in contexts of permanently faulty nodes has been studied in a number of works [8, 13, 16, 17, 18], but only assuming that the faulty nodes are chosen by an adversary. The difference between such worst case scenarios and the probabilistic version studied here is again significant, both in terms of results and techniques (see more details in Section 1.3).

1.1 The Noisy Advice Model

We start with some notation. Further notations are given in Section 1.4. Let T be an n-node tree[1] rooted at some arbitrary node σ. We consider an agent that is initially located at the root σ of T, aiming to find a node τ, called the *treasure*, which is chosen by an adversary. The distance $d(u, v)$ is the number of edges on the path between u and v. The depth of a node u is $d(u) = d(\sigma, u)$. Let $d = d(\tau)$ denote the depth of τ, and let the depth D of the tree be the maximal depth of a node. Finally, let Δ_u denote the degree of node u and let Δ denote the maximal degree in the tree.

Each node $u \neq \tau$ is assumed to be provided with an *advice*, termed $\texttt{adv}(u)$, which provides information regarding the direction of the treasure. Specifically, $\texttt{adv}(u)$ is a pointer to one of u's neighbors. It is called *correct* if the pointed neighbor is one step closer to the treasure than u is. Each vertex $u \neq \tau$ is *faulty* with probability q_u (the meaning of being faulty will soon be explained). Otherwise, u is considered *sound*, in which case its advice is correct. We call q_u the *noise parameter* of u, and define the *general noise parameter* as $q = \max\{q_u \mid u \in T\}$.

We consider two models for faulty nodes. The main model assumes that the advice at a faulty node points to one of its neighbors chosen uniformly at random (and so possibly pointing at the correct one). We also consider an adversarial variant, called the *semi-adversarial model*, where this neighbor is chosen by an oblivious adversary. That is, an adversary specifies for each node what advice it would have assuming it is faulty. Then, faulty nodes are still chosen randomly as in the main model, but their advice is specified by the adversary.

The agent can move by traversing edges of the tree. At any time, the agent can query its hosting node in order to "see" the corresponding advice and to detect whether the treasure is present there. The protocol terminates when the agent queries the treasure. We evaluate a search algorithm $\texttt{A}$ by two measures: The move complexity, termed $\mathcal{M}(\texttt{A})$, is the expected number of edge traversals, and the query complexity, termed $\mathcal{Q}(\texttt{A})$, is the expected number

[1] We present the model for trees, but it should be clear that it can be similarly defined for arbitrary graphs (see also Section 5).

of queries[2]. Expectation is taken over both the randomness involved in sampling advice and the possible probabilistic choices made by $\mathtt{A}$. We note that when considering walking algorithms, we assume that the agent does not know the structure of the tree in advance, and discovers it as it moves. Conversely, when focusing on minimizing the query complexity only, we assume that the tree structure is known to the algorithm.

The noise parameters $(q_u)_{u \in T}$ govern the accuracy of the environment. On the one extreme, if $q_u = 0$ for all nodes, then advice is always correct. This case allows to find the treasure in d moves, by simply following each encountered advice. Alternatively, it also allows to find the treasure using $\mathcal{O}(\log n)$ queries, by performing a separator based search. On the other extreme, if $q_u = 1$ for all nodes, then advice is essentially meaningless, and the search cannot be expected to be efficient. An intriguing question is therefore to identify the largest value of q that allows for efficient search.

1.2 Our Results

Consider the noisy advice model on trees with maximum degree Δ and depth D. Roughly speaking, we show that $1/\sqrt{\Delta}$ is the threshold for the noise parameter q, in order to obtain search algorithms with low expected complexities.

The proof that there is no algorithm with low expected complexities when the noise exceeds $1/\sqrt{\Delta}$ is rather simple, and in fact, holds even if the algorithm has access to the advice of all internal nodes. Intuitively, the argument is as follows (the formal proof appears in Section 4.1). Consider a complete Δ-ary tree of depth D and assume that the treasure τ is placed at a leaf. The first observation (Lemma 10) is that the expected number of leaves having more advice point to them than to τ is a lower bound on the query complexity. The next observation is that there are roughly Δ^D leaves whose distance from τ is $2D$. For each of those leaves u, the probability that more advice points towards it than towards τ can be approximated by the probability that all nodes on path connecting u and τ are faulty. As this latter probability is q^{2D}, the expected number of leaves that have more pointers leading to them is roughly $\Delta^D q^{2D}$, which explodes when $q \gg 1/\sqrt{\Delta}$. This essentially establishes the lower bound for the noise regime.

The main technical difficulties we had to face appeared when we aimed to show that the $1/\sqrt{\Delta}$ lower bound is, in fact, tight, and moreover, that there exist extremely efficient algorithms when the noise is above the threshold. In this regard, we note two technical contributions. The first appears in the construction of the moving algorithm $\mathtt{A}_{walk}$. Even though the algorithm should be designed to quickly find an adversarially placed treasure, it is in fact based on a Bayesian approach. The challenging part is identifying the correct prior. Constructing algorithms that ensure worst-case guarantees through a Bayesian approach was done in [4] which studies a closely related, yet much simpler problem of search on the line. Apart from [4] we are unaware of other works that follow this approach. The second technical contribution corresponds to the query setting, where we mimic the resampling of advice at separator nodes, by locally applying the moving algorithm.

1.2.1 Upper Bounds

In Section 2, we present a walking algorithm that is optimal up to a constant factor for the regime of noise below the threshold. Furthermore, this algorithm does not require prior knowledge of either the tree's structure, or the values of Δ, q, d, or n.

[2] The success probability after a fixed number of rounds is another quantity of interest. It is left for future work.

Using this walking algorithm, we derive two query algorithms (in Section 3). The first is optimal up to a factor of $\mathcal{O}(\log^2(\Delta)\log n)$ and the second is restricted to regular trees, but is optimal up to a factor of $\mathcal{O}(\log(\Delta)\log\log n)$. Note that the query algorithms use the knowledge of the tree structure, as well as bounds on the regime of noise.

Before stating our theorems, we need the following definition.

▶ **Definition 1.** Condition ($\star$) holds with parameter $0 < \varepsilon < 1$ if for every node v, we have

$$q_v < \frac{1 - \varepsilon - \Delta_v^{-\frac{1}{4}}}{\sqrt{\Delta_v} + \Delta_v^{\frac{1}{4}}}.$$

Note that since $\Delta_v \geq 2$, the condition is always satisfiable when taking a small enough ε. In the following theorems the $\mathcal{O}$ notation hides only a polynomial a polynomial term in $1/\varepsilon$.

Note, all our algorithms are deterministic, hence, expectation is taken with respect only to the sampling of the advice.

▶ **Theorem 2.** *There exists a deterministic walking algorithm $\mathtt{A}_{walk}$ such that for any constant $\varepsilon > 0$, if Condition ($\star$) holds with parameter ε then $\mathcal{M}(\mathtt{A}_{walk}) = \mathcal{O}(\sqrt{\Delta}d)$.*

▶ **Theorem 3.**

1. *For any $\varepsilon > 0$, there exists a deterministic query algorithm $\mathtt{A}_{sep}$ such that if Condition ($\star$) holds with parameter ε then the query complexity is $\mathcal{Q}(\mathtt{A}_{sep}) = \mathcal{O}(\sqrt{\Delta}\log\Delta \cdot \log^2 n)$.*
2. *Assume that $q < c/\sqrt{\Delta}$ for a small enough constant $c > 0$. Then there exists a deterministic query algorithm $\mathtt{A}_{2-layers}$ such that, restricted to (not necessarily complete) Δ-ary trees, $\mathcal{Q}(\mathtt{A}_{2-layers}) = \mathcal{O}(\sqrt{\Delta}\log n \cdot \log\log n)$.*

1.2.2 Lower Bounds

We establish the following lower bound. The main part of the proof is to be found in Section 4. We refer the reader to the full version of this work [5, Section 2 and Appendix A] for the missing parts.

▶ **Theorem 4.** *The following holds for any randomized algorithm $\mathtt{A}$ and any integer $\Delta \geq 3$.*

1. ***Exponential complexity above the threshold.***
Consider a complete Δ-ary tree. For every constant $\varepsilon > 0$, if $q \geq \frac{1+\varepsilon}{\sqrt{\Delta-1}} \cdot (1 + \frac{1}{\Delta-1})$, then both $\mathcal{Q}(\mathtt{A})$ and $\mathcal{M}(\mathtt{A})$ are exponential in D.
2. ***Lower bounds for any q.***
(a) Consider a complete Δ-ary tree. Then $\mathcal{Q}(\mathtt{A}) = \Omega(q\Delta\log_\Delta n)$.
(b) For any integer d, there is a tree with at most $d\Delta$ nodes, and a placement of the treasure at depth d, such that $\mathcal{M}(\mathtt{A}) = \Omega(dq\Delta)$.

Observe that taken together, Theorems 2,4,3 and Condition ($\star$) imply that for any $\varepsilon > 0$ and large enough Δ, efficient search can be achieved if $q < (1-\varepsilon)/\sqrt{\Delta}$ but not if $q > (1+\varepsilon)/\sqrt{\Delta}$.

1.2.3 Memory-less Algorithms

Query algorithms assume the knowledge of the tree and hence cannot avoid memory complexity which is linear in n. In contrast, our walking algorithm $\mathtt{A}_{walk}$ uses memory that is composed of advice accumulated during the walk, and hence remains low, in expectation.

Finally, we analyse the performance of simple memoryless algorithms called *probabilistic following*, suggested in [15]. At every step, the algorithm follows the advice at the current vertex with some fixed probability λ, and performs a random walk step otherwise. It turns out that such algorithms can perform well, but only in a very limited regime of noise. Specifically, we prove:

▶ **Theorem 5.** *There exist positive constants c_1, c_2 and c_3 such that the following holds. If for every vertex u, $q_u < c_1/\Delta_u$ then there exists a probabilistic following algorithm that finds the treasure in less than $c_2 d$ expected steps. On the other hand, if $q > c_3/\Delta$ then for any probabilistic following strategy the move complexity on a complete Δ-ary tree is exponential in the depth of the tree.*

Since this algorithm is randomized, expectation is taken over both the randomness involved in sampling advice and the possible probabilistic choices made by the algorithm.

Interestingly, when $q_u < c_1/\Delta_u$ for all vertices, this algorithm works even in a semi-adversarial model. In fact, it turns out that in the semi-adversarial model, probabilistic following algorithms are the best possible, as the threshold for efficient search, with respect to any algorithm, is roughly $1/\Delta$. Due to lack of space these results are discussed and proved in the full version of this work [5, Appendix E].

1.3 Related Work

In computer science, search algorithms have been the focus of numerous works. Due to their importance, trees are particularly popular structures to investigate search, see e.g., [20, 3, 22, 21]. Within the literature on search, many works considered noisy queries [12, 19, 11], however, it was typically assumed that noise can be *resampled* at every query. As mentioned, dealing with permanent faults incurs challenges that are fundamentally different from those that arise when allowing queries to be resampled. To illustrate this difference, consider the simple example of a star graph and a constant $q < 1$. Straightforward amplification can detect the target in $\mathcal{O}(1)$ expected number of queries. In contrast, in our model, it can be easily seen that $\Omega(n)$ is a lower bound for both the move and the query complexities, for any constant noise parameter.

A search problem on graphs in which the set of nodes with misleading advice is chosen by an adversary was studied in [16, 17, 18], as part of the more general framework of the *liar models* [1, 2, 6, 9, 23]. Data structures with adversarial memory faults have been investigated in the so called faulty-memory RAM model introduced in [14]. In particular, data structures supporting the same operations as search trees with adversarial memory faults were studied in [13, 8]. Interestingly, the data structures developed in [8] can cope with up to $O(\log n)$ faults, happening at any time during the execution of the algorithm, while maintaining optimal space and time complexity. All these worst case models are, however, significantly different from the randomized one we consider, both in terms of techniques and results. The subject of queries with probabilistic memory faults, as the ones we study here, has been explicitly studied in the context of sorting [7].

The noisy advice model considered in this paper actually originated in the recent biologically centered work [15], aiming to abstract navigation relying on guiding instructions in the context of collaborative transport by ants. There, a group of ants carry a large load of food aiming to transport it to their nest, while basing their navigation on unreliable advice given by pheromones that are laid on the terrain. In that work, the authors modelled ant navigation as a probabilistic following algorithm, and noticed that an execution of such an algorithm can be viewed as an instance of Random Walks in Random Environment (RWRE) [24, 10]. Relying on results from this subfield of probability theory, the authors showed that when tuned properly, such algorithms enjoy linear move complexity on grids, provided that the bias towards the correct direction is sufficiently high.

1.4 Notations

Denote $p = 1 - q$, and for a node u, $p_u = 1 - q_u$. For two nodes u, v, let $\langle u, v\rangle$ denote the simple path connecting them, excluding the end nodes, and let $[u, v\rangle = \langle u, v\rangle \cup \{u\}$ and $[u, v] = [u, v\rangle \cup \{v\}$. For a node u, let $T(u)$ be the subtree rooted at u. We denote by $\overrightarrow{\texttt{adv}}(u)$ (resp. $\overleftarrow{\texttt{adv}}(u)$) the set of nodes whose advice points towards (resp. away from) u. By convention $u \notin \overrightarrow{\texttt{adv}}(u) \cup \overleftarrow{\texttt{adv}}(u)$. Unless stated otherwise, log is the natural logarithm.

1.5 Organization

In Section 2 we present our optimal walking algorithm. Section 3 presents our query algorithms, while most of the details regarding the more elaborated algorithm on regular trees are only shown in the full version of this work [5, Appendix G]. In Section 4 we show the lower bounds for both the move and query complexities. In Section 5, we give a list of open problems. Theorem 5 and the threshold of $\Theta(1/\Delta)$ that applies to the semi-adversarial setting are proved in the full version of this work.

2 Optimal Walking Algorithm

In this section we prove Theorem 2. At a very high level, at any given time, the walking algorithm processes the advice seen so far, identifies a promising node to continue from on the border of the already discovered connected component, moves to that node, and explores one of its neighbors.

2.1 Algorithm Design following a Greedy Bayesian Approach

In our setting the treasure is placed by an adversary. However, we can still study algorithms induced by assuming that it is placed in one of the vertices according to some known distribution and see how they measure up in our worst case setting. As mentioned, this approach is similar to [4], which studies the closely related, yet much simpler problem of search on the line. Of course, the success of this scheme highly depends on the choice of the prior distribution we take.

To make our life easier, let us first assume that the structure of the tree is known to the algorithm. Also, we assume the treasure is placed in one of the leaves of the tree according to some known distribution θ, and denote by `adv` the advice on the nodes we have already visited. Aiming to find the treasure as fast as possible, a possible greedy algorithm explores the vertex that, given the advice seen so far, has the highest probability of having the treasure in its subtree.

We extend the definition of θ to internal nodes by defining $\theta(u)$ to be the sum of $\theta(w)$ over all leaves w of $T(u)$. Given some u that was not visited yet, and given the previously seen advice `adv`, the probability of the treasure being in u's subtree $T(u)$, is:

$$\begin{aligned}\mathbb{P}\left(\tau \in T(u) \,|\, \texttt{adv}\right) &= \frac{\mathbb{P}\left(\tau \in T(u)\right)}{\mathbb{P}\left(\texttt{adv}\right)}\mathbb{P}\left(\texttt{adv} \,|\, \tau \in T(u)\right)\\ &= \frac{\theta(u)}{\mathbb{P}\left(\texttt{adv}\right)} \prod_{w \in \overrightarrow{\texttt{adv}}(u)} \left(p_w + \frac{q_w}{\Delta_w}\right) \prod_{w \in \overleftarrow{\texttt{adv}}(u)} \frac{q_w}{\Delta_w}.\end{aligned}$$

The last factor is q_w/Δ_w because it is the probability that the advice at w points exactly the way it does in `adv`, and not only away from τ. Note that the advice seen so far is never for vertices in $T(u)$ as we consider a walking algorithm, and u has not been visited yet. Therefore, if $\tau \in T(u)$ then correct advice in `adv` points to u. We ignore the term $p_w + q_w/\Delta_w$ as it is normally quite close to 1, and applying a log we can approximate the relative strength of a node by:

$$\log(\theta(u)) + \sum_{w \in \overleftarrow{\mathtt{adv}}(u)} \log\left(\frac{q_w}{\Delta_w}\right).$$

We do not want to assume that our algorithm knows q_w, but we do assume that in the worst scenario $q_w \sim 1/\sqrt{\Delta_w}$. Assigning this value and rescaling we finally define:

$$\mathtt{score}(u) = \frac{2}{3}\log(\theta(u)) - \sum_{w \in \overleftarrow{\mathtt{adv}}(u)} \log(\Delta_w).$$

When comparing two specific vertices u and v, $\mathtt{score}(u) > \mathtt{score}(v)$ iff:

$$\sum_{w \in \langle u,v\rangle \cap \overrightarrow{\mathtt{adv}}(u)} \log(\Delta_w) - \sum_{w \in \langle u,v\rangle \cap \overrightarrow{\mathtt{adv}}(v)} \log(\Delta_w) > \frac{2}{3}\log\left(\frac{\theta(v)}{\theta(u)}\right).$$

This is because any advice that is not on the path between u and v contributes the same to both sides, as well as advice on vertices on the path that point sideways, and not towards u or v[3]. Since we use this score to compare two vertices that are neighbors of already explored vertices, and our algorithm is a walking algorithm, then we will always have all the advice on this path. In particular, the answer to whether $\mathtt{score}(u) > \mathtt{score}(v)$, does not depend on the specific choices of the algorithm, and does not change throughout the execution of the algorithm, even though the scores themselves do change. The comparison depends only on the advice given by the environment.

Let us try and justify the score criterion at an intuitive level. Consider the case of a complete Δ-ary tree, with θ being the uniform distribution on the leaves[4]. Here $score(u) > score(v)$ if (cheating a little by thinking of $\log(\Delta)$ and $\log(\Delta - 1)$ as equal):

$$\left|\overrightarrow{\mathtt{adv}}(u) \cap \langle u,v\rangle\right| - \left|\overrightarrow{\mathtt{adv}}(v) \cap \langle u,v\rangle\right| > \frac{2}{3}\big(d(u) - d(v)\big).$$

If, for example, we consider two vertices $u, v \in T$ at the same depth, then $score(u) > score(v)$ if there is more advice pointing towards u than towards v. If the vertices have different depths, then the one closer to the root has some advantage, but it can still be beaten.

For general trees, perhaps the most natural θ to take is the uniform distribution on all nodes (or just on all leaves - this choice is actually similar). It is also a generalization of the example above. Unfortunately, however, while this works well on the complete Δ-ary tree, we show in the full version of this paper [5, Appendix D], that this approach fails on other (non-complete) Δ-ary trees.

2.2 Algorithm $\mathtt{A}_{walk}$

In our context, there is no distribution over treasure location and we are free to choose θ as we like. We take θ to be the distribution defined by a simple random process. Starting at

[3] It is tempting to define $\mathtt{score}(u)$ as the sum of weighted advice from the root to u. However, when comparing two vertices, the advice of their least common ancestor would be counted twice, which we prefer to avoid.

[4] Actually, a similar formula could be derived choosing θ to be the uniform distribution over all nodes, but for technical reasons it is easier to restrict it to leaves only.

the root, at each step, walk down to a child uniformly at random. until reaching a leaf. For a leaf v, define $\theta(v)$ as the probability that this process eventually reaches v. Our extension of θ can be interpreted as $\theta(v)$ being the probability that this process passes through v. Formally, $\theta(\sigma) = 1$, and $\theta(u) = (\Delta_\sigma \prod_{w\in\langle\sigma,u\rangle}(\Delta_w - 1))^{-1}$. It turns out that this choice, slightly changed, works remarkably well, and gives an optimal algorithm in noise conditions that practically match those of our lower bound. For a vertex $u \neq \sigma$, define:

$$\beta(u) = \prod_{w\in[\sigma,u\rangle} \Delta_w.$$

It is a sort of approximation of $1/\theta(u)$, which we prefer for technical convenience. Indeed, for all u, $1/\beta(u) \leq \theta(u)$. A wonderful property of this β (besides the fact that it gives rise to an optimal algorithm) is that to calculate $\beta(v)$ (just like θ), one only needs to know the degrees of the vertices from v up to the root. It is hard to imagine distributions on leaves that allow us to calculate the probability of being in a subtree without knowing anything about it!

In the walking algorithm, if v is a candidate for exploration, these nodes must have been visited already and so the algorithm does not need any a priori knowledge of the structure of the tree. The following claim will be soon useful:

▶ **Claim 6.** *The following two inequalities hold for every $c < 1$:*

$$\sum_{v\in T} \frac{c^{d(v)}}{\beta(v)} \leq \frac{1}{1-c}, \quad \sum_{v\in T} \frac{d(v)c^{d(v)}}{\beta(v)} \leq \frac{c}{(1-c)^2}.$$

Proof. To prove the first inequality, follow the same random walk defining θ. Starting at the root, at each step choose uniformly at random one of the children of the current vertex. Now, while passing through a vertex v collect $c^{d(v)}$ points. No matter what choices are made, the number of points is at most $1 + c + c^2 + ... = 1/(1-c)$. On the other hand, $\sum_{v\in T}\theta(v)c^{d(v)}$ is the expected number of points gained. The result follows since $1/\beta(v) \leq \theta(v)$. The second inequality is derived similarly, using the fact that $c + 2c^2 + 3c^3 + \ldots = c/(1-c)^2$. ◀

For a vertex $u \in T$ and previously seen advice `adv` define:

$$\texttt{score}(u) = \frac{2}{3}\log\left(\frac{1}{\beta(u)}\right) - \sum_{w\in\overleftarrow{\texttt{adv}}(u)} \log(\Delta_w).$$

Algorithm $\texttt{A}_{walk}$ keeps track of all vertices that are children of the vertices it explored so far, and repeatedly walks to and then explores the one with highest score according to the current advice, breaking ties arbitrarily. Note that the algorithm does not require prior knowledge of either the tree's structure, or the values of Δ, q, d or n.

2.3 Analysis

Recall the definition of Condition ($\star$) from Definition 1. The next lemma provides a large deviation bound tailored to our setting. The proof can be found in Appendix C of the full version [5].

▶ **Lemma 7.** *Consider independent random variables $X_1, \ldots, X_\ell$, where X_i takes the values $(-\log\Delta_i, 0, \log\Delta_i)$ with respective probabilities $(p_i + \frac{q_i}{\Delta_i}, q_i(1-\frac{2}{\Delta_i}), \frac{q_i}{\Delta_i})$, for parameters $p_i, q_i = 1 - p_i$ and $\Delta_i > 0$. Assume that Condition ($\star$) holds for some $\varepsilon > 0$. Then for every integer (positive or negative) m,*

$$\mathbb{P}\left(\sum_{i=1}^{\ell} X_i \geq m\right) \leq \frac{(1-\varepsilon)^\ell}{e^{\frac{3m}{4}}}\prod_{i=1}^{\ell}\frac{1}{\sqrt{\Delta_i}}.$$

The next theorem states that $\mathtt{A}_{walk}$ is optimal up to a constant factor for the regime of noise below the threshold. It establishes Theorem 2.

▶ **Theorem 8.** *Assume that Condition* $(\star)$ *holds for some fixed* $\varepsilon > 0$*. Then* $\mathcal{M}(\mathtt{A}_{walk}) = \mathcal{O}(d\sqrt{\Delta})$*, where the constant hidden in the* $\mathcal{O}$ *notation only depends polynomially on* $1/\varepsilon$*.*

Proof. Denote the vertices on the path from σ to τ by $\sigma = u_0, u_1, \ldots, u_d = \tau$ in order. Denote by E_k the expected time to reach u_k once u_{k-1} is reached. We will show that for all k, $E_k = \mathcal{O}(\sqrt{\Delta})$, and by linearity of expectation this concludes the proof.

Once u_{k-1} is visited, $\mathtt{A}_{walk}$ only goes to some of the nodes that have score at least as high as u_k. We can therefore bound E_k from above by assuming we go through all of them, and this expression does not depend on the previous choices of the algorithm and the nodes it saw before seeing u_k. The length of this tour is bounded by twice the sum of distances of these nodes from u_k. Hence,

$$E_k \le 2\sum_{i=1}^{k} \sum_{u \in C(u_i)} \mathbb{P}\left(\mathtt{score}(u) \ge \mathtt{score}(u_k)\right) \cdot d(u_k, u).$$

Where $C(u_k) = T(u_{k-1}) \setminus T(u_k)$, and so $\cup_{i=1}^{k} C(u_i) = T \setminus T(u_k)$. Recall that scores are defined so that u has a larger score than u_k, if the sum of weighted arrows on the path $\langle u_k, u\rangle$ is at least $\frac{2}{3}\log(\beta(u)/\beta(u_k))$. Setting m to be this value, Lemma 7 allows to calculate this probability exactly. Indeed, a vertex x on the path should point towards u_k: this happens with probability $p_x + q_x/\Delta_x$. Otherwise, it points towards u with probability q_x/Δ_x, and elsewhere with probability $q_x(1 - 2/\Delta_x)$. Denoting $c = 1 - \varepsilon$,

$$\begin{aligned}
\frac{E_k}{2} &\le \sum_{i=1}^{k} \sum_{u \in C(u_i)} \frac{c^{d(u_k,u)-1}}{e^{\frac{3}{4}\cdot\frac{2}{3}\log\left(\frac{\beta(u)}{\beta(u_k)}\right)}} \sqrt{\prod_{v \in \langle u, u_k\rangle} \frac{1}{\Delta_v}} \cdot d(u_k, u) \\
&= \frac{1}{c}\sum_{i=1}^{k} \sum_{u \in C(u_i)} \frac{c^{d(u_k,u)}}{\sqrt{\frac{\beta(u)}{\beta(u_k)}}} \sqrt{\frac{\Delta_{u_i}}{\frac{\beta(u_k)}{\beta(u_i)} \cdot \frac{\beta(u)}{\beta(u_i)}}} \cdot d(u_k, u) \\
&\le \frac{\sqrt{\Delta}}{c} \sum_{i=1}^{k} c^{d(u_k,u_i)} \sum_{u \in C(u_i)} c^{d(u_i,u)} \frac{\beta(u_i)}{\beta(u)} \cdot \Big(d(u_k, u_i) + d(u_i, u)\Big).
\end{aligned}$$

By Claim 6, applied to the tree rooted at u_i, we get:

$$\sum_{u \in C(u_i)} \frac{c^{d(u_i,u)}\beta(u_i)}{\beta(u)} < \frac{1}{1-c}, \quad \text{and} \quad \sum_{u \in C(u_i)} \frac{c^{d(u_i,u)}\beta(u_i)}{\beta(u)} d(u_i, u) < \frac{c}{(1-c)^2}.$$

And so:

$$\begin{aligned}
\frac{E_k}{2} &\le \frac{\sqrt{\Delta}}{c(1-c)} \sum_{i=1}^{k} c^{d(u_k,u_i)} d(u_k, u_i) + \frac{\sqrt{\Delta}}{(1-c)^2} \sum_{i=1}^{k} c^{d(u_k,u_i)} \\
&\le \frac{(1+c)\sqrt{\Delta}}{(1-c)^3} \le \frac{2\sqrt{\Delta}}{\varepsilon^3} = \mathcal{O}\left(\sqrt{\Delta}\right),
\end{aligned}$$

where we again used the equality $c + 2c^2 + 3c^3 + \ldots = c/(1-c)^2$. ◀

3 Query Algorithms

3.1 An $\mathcal{O}(\sqrt{\Delta}\log\Delta\log^2 n)$ Queries Algorithm

Our next goal is to prove the first item in Theorem 3. As is common in search on trees, our technique in this section is based on separators. We say a node u is a *separator* of T if all the connected components of $T\setminus\{u\}$ are of size at most $|T|/2$. It is well known that such a node exists. Assume there is some local procedure, that given a vertex u decides with probability $1-\delta$ in which one of the connected components of $T\setminus\{u\}$, the treasure resides. Applying this procedure on a separator of the tree, and then focusing the search recursively only on the component it pointed out, results in a type of algorithm we call a *separator based* algorithm. It uses the local procedure at most $\lceil\log_2 n\rceil$ times, and by a union bound, finds the treasure with probability at least $1-\lceil\log_2 n\rceil\delta$. Broadly speaking, we will be interested in the expected running time of this sort of algorithm conditioned on it being successful. This sort of conditioning complicates matters slightly. In what follows, we assume that the set of separators for the tree is fixed.

Proof. *(of the first item in Theorem 3)* The algorithm we build is denoted $\mathtt{A}_{sep}$. It runs a separator based algorithm in parallel to some arbitrary exhaustive search algorithm. The meaning of *in parallel* here simply means that the two algorithms are run in an alternating fashion. Fix some small h. The local exploration procedure, denoted $\mathtt{local}_h$, for a vertex u proceeds as follows.

Procedure $\mathtt{local}_h(u)$. Consider the tree $T_h(u)$ rooted at u consisting of all vertices satisfying $\log_\Delta\beta(v)<h$ together with their children. So a leaf of $v\in T_h(u)$ is either a leaf of T, or satisfies $\Delta^h\le\beta(v)<\Delta^{h+1}$. Denote the second kind a *nominee*. Call a nominee *promising* if the number of weighted arrows pointing to v is large, specifically, if $\sum_{w\in[u,v\rangle}X_w\ge\frac{2}{3}h\log\Delta$, where $X_w=\log\Delta_w$ if the advice at w is pointing to v, $X_w=-\log\Delta_w$ if it is pointing to u, and $X_w=0$ otherwise. Viewing it as a query algorithm, we now run the walking algorithm $\mathtt{A}_{walk}$ on $T_h(u)$ (starting at its root u) until it either finds the treasure or finds a promising nominee. In the latter case, $\mathtt{local}_h(u)$ declares that the treasure is on the connected component of $T\setminus\{u\}$ containing this nominee. If $\tau\in T_h(u)$ then set $\tau_u=\tau$. Otherwise let τ_u be the leaf of $T_h(u)$ closest to the treasure, and so in this case τ_u is a nominee. Say that u is *h-misleading* if either (1) $\tau\notin T_h(u)$ and τ_u is not promising, or (2) there is some promising nominee $v\in T_h(u)$ that is not in the same connected component of $T\setminus\{u\}$ as τ_u. Note that if u is not h-misleading then $\mathtt{local}_h(u)$ necessarily outputs the correct component of $T\setminus\{u\}$, namely, the one containing the treasure. The proof of the following lemma appears in the full version of this work, [5, Appendix F]. The part regarding regular trees will be needed later.

▶ **Lemma 9.** *For any u, $\mathbb{P}(u$ is h-misleading$)\le(\Delta+1)(1-\varepsilon)^h$. Also, for any event X such that X occurring always implies that u is not misleading, we have $\mathbb{P}(X)\,\mathcal{Q}(\mathtt{local}_h(u)\mid X)=\mathcal{O}(\sqrt{\Delta}\log\Delta\cdot h)$. In the case the tree is regular, these bounds become $2(1-\varepsilon)^h$ and $\mathcal{O}(\sqrt{\Delta}\cdot h)$ respectively. The constant hidden in the $\mathcal{O}$ notation only depends polynomially on $1/\varepsilon$.*

Taking $h=-3\log(2n)/\log(1-\varepsilon)$, gives $\mathbb{P}(u\text{ is misleading})\le 1/n^2$. Denote by $\mathtt{Good}$ the event that none of the separators encountered are misleading. By a union bound, $\mathbb{P}(\mathtt{Good}^c)\le 1/n$.

$$\mathcal{Q}(\mathtt{A}_{sep})=\mathbb{P}(\mathtt{Good})\,\mathcal{Q}(\mathtt{A}_{sep}\mid\mathtt{Good})+\mathbb{P}(\mathtt{Good}^c)\,\mathcal{Q}(\mathtt{A}_{sep}\mid\mathtt{Good}^c)\,. \tag{1}$$

As $\mathtt{A}_{sep}$ runs an exhaustive search algorithm in parallel, the second term is $\mathcal{O}(1)$. For the first term, note that conditioning on $\mathtt{Good}$, all local procedures either find the treasure or give the correct answer, and so there are $\mathcal{O}(\log n)$ of them and they eventually find the treasure. Denote by u_i the i-th vertex that $\mathtt{local}_h$ is executed on. By linearity of expectation, and applying Lemma 9, the first term of (1) is $\mathbb{P}(\mathtt{Good}\ )\sum_i \mathcal{Q}(\mathtt{local}_h(u_i) \mid \mathtt{Good}\ ) = \mathcal{O}(\log n \cdot \sqrt{\Delta}\log\Delta \cdot h) = \mathcal{O}(\sqrt{\Delta}\log\Delta\log^2 n)$. As $\log(1+x) > x$ always, then $-1/\log(1-\varepsilon) \leq 1/\varepsilon$, and the hidden factor in the $\mathcal{O}$ is as stated. ◀

3.2 An Almost Tight Result for Regular Trees

We now turn our attention to the second item in Theorem 3. Due to space constraints, we only sketch the argument here and refer the interested reader to the full version for details. At a high level, we run two algorithms in parallel (i.e., in an alternating fashion): $\mathtt{A}_{fast}$, and $\mathtt{A}_{mid}$. Algorithm $\mathtt{A}_{fast}$ is actually $\mathtt{A}_{sep}$ applied with parameter $h = \Theta(\log\log n)$ instead of $\Theta(\log n)$. Using Lemma 9, with probability $1 - 1/\log^{\mathcal{O}(1)}(n)$, the local procedure of $\mathtt{A}_{fast}$ always detects the correct component for each separator, and $\mathtt{A}_{fast}$ needs an expected number of $\mathcal{O}(\sqrt{\Delta} \cdot \log n \cdot \log\log n)$ queries to find the treasure. This is the running time we are aiming for.

Algorithm $\mathtt{A}_{mid}$ is similar to $\mathtt{A}_{sep}$ except it uses a different subroutine for local exploration. It then remains to show that it finds the treasure using a relatively low expected number of queries even conditioning on the event that caused $\mathtt{A}_{fast}$ to fail, namely, the event that there is a misleading separator at the scale $h = \Theta(\log\log n)$. The query complexity of $\mathtt{A}_{mid}$ does blow up under this event but we show that the blowup is not that bad, and can be compensated by the fact that the bad event has small probability. This is the core of the proof, and what requires most work. In fact, the complexity of the arguments led us to restrict the discussion to regular trees and also modify the subroutine for local exploration to ease the analysis.

4 Lower Bounds

We next prove Items (1) of Theorem 4. Items (2a) and (2b) are proved in Appendix A of the full version of this paper.

4.1 Exponential Complexity Above the Threshold

We wish to prove Item (1) in Theorem 4. Namely, that for every fixed $\varepsilon > 0$, and for every complete Δ-ary tree, if $q \geq \frac{1+\varepsilon}{\sqrt{\Delta-1}} \cdot (1 + \frac{1}{\Delta-1})$, then every randomized search algorithm has query (and move) complexity which is both exponential in the depth d of the treasure and polynomial in n. In fact, this lower bound holds even if the algorithm has access to the advice of all internal nodes. The following lemma is proved in stated here without proof:

▶ **Lemma 10.** *Assume the treasure is placed in a leaf τ of the complete Δ-ary tree. Denote by* $\mathtt{adv}$ *the random advice on all internal nodes, then the expected number of leaves u satisfying $|\overrightarrow{\mathtt{adv}}(u)| > |\overrightarrow{\mathtt{adv}}(\tau)|$, is a lower bound on the query complexity of any algorithm.*

Using Lemma 10, all we need to do is approximate the number of leaves u satisfying $|\overrightarrow{\mathtt{adv}}(u)| > |\overrightarrow{\mathtt{adv}}(\tau)|$. When comparing the number of pointers that point towards each of two different nodes, only the pointers of the internal nodes on the path between them may influence on the result. The probability that a leaf u "beats" the treasure τ in the sense of

Lemma 10, is at least the probability that exactly one node on the path points to u and none of the rest point towards the treasure. This probability is at least

$$\frac{q}{\Delta} \cdot \left(q \cdot \left(1 - \frac{1}{\Delta}\right)\right)^{d(u,\tau)-2}.$$

There are precisely $(\Delta - 1)^D$ leaves whose distance from the treasure is $2D$. Therefore, the expected number of leaves that beat the treasure is at least:

$$\begin{aligned}\frac{q}{\Delta}(\Delta-1)^D q^{2D-2} \cdot \left(1 - \frac{1}{\Delta}\right)^{2D-2} &= \frac{\Delta}{q(\Delta-1)^2} \cdot \left(\frac{q^2(\Delta-1)^3}{\Delta^2}\right)^D \\ &\geq \frac{\Delta}{q(\Delta-1)^2} \cdot (1+\varepsilon)^2 D.\end{aligned}$$

Item (1) in Theorem 4 follows.

5 Open Problems

Closing the small gap between the upper and lower bounds for the query setting remains open. The noisy advice model may well be interesting to study in other search settings. In particular, obtaining efficient search algorithms for general graphs is highly intriguing. Even though the likelihood of a node being the treasure under a uniform prior can still be computed in principle, it is not so easy to compare two nodes as in Theorem 8 because there may be more than a single path between them.

In a limited regime of noise, we believe that memoryless strategies might very well be efficient also on general graphs, and we pose the following conjecture. Proving it may require the use of tools from the theory of RWRE, which seem to be lacking in the context of general graph topologies.

▶ **Conjecture 11.** *There exists a probabilistic following algorithm that finds the treasure in expected linear time on any undirected graph assuming $q < c/\Delta$ for a small enough $c > 0$.*

References

1 Andrei Asinowski, Jean Cardinal, Nathann Cohen, Sébastien Collette, Thomas Hackl, Michael Hoffmann, Kolja B. Knauer, Stefan Langerman, Michal Lason, Piotr Micek, Günter Rote, and Torsten Ueckerdt. Coloring hypergraphs induced by dynamic point sets and bottomless rectangles. *CoRR*, abs/1302.2426, 2013. URL: `http://arxiv.org/abs/1302.2426`.

2 Javed A. Aslam and Aditi Dhagat. Searching in the presence of linearly bounded errors. In *STOC*, pages 486–493. ACM, 1991. [doi:10.1145/103418.103469,].

3 Yosi Ben-Asher, Eitan Farchi, and Ilan Newman. Optimal search in trees. *SIAM J. Comput.*, 28(6):2090–2102, 1999. [doi:10.1137/S009753979731858X,].

4 Michael Ben-Or and Avinatan Hassidim. The bayesian learner is optimal for noisy binary search (and pretty good for quantum as well). In *FOCS*, pages 221–230, 2008. [doi:10.1109/FOCS.2008.58,].

5 Lucas Boczkowski, Amos Korman, and Yoav Rodeh. Searching a tree with permanently noisy advice. *https://arxiv.org/abs/1611.01403*, 2016. [arXiv:1611.01403,].

6 Ryan S. Borgstrom and S. Rao Kosaraju. Comparison-based search in the presence of errors. In *STOC*, pages 130–136. ACM, 1993.

7 Mark Braverman and Elchanan Mossel. Noisy sorting without resampling. In *SODA*, pages 268–276, 2008. URL: `http://dl.acm.org/citation.cfm?id=1347082.1347112`.

8 Gerth Stølting Brodal, Rolf Fagerberg, Irene Finocchi, Fabrizio Grandoni, Giuseppe F. Italiano, Allan Grønlund Jørgensen, Gabriel Moruz, and Thomas Mølhave. Optimal resilient dynamic dictionaries. In *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 347–358, 2007. [doi:10.1007/978-3-540-75520-3_32,].

9 Ferdinando Cicalese and Ugo Vaccaro. Optimal strategies against a liar. *Theor. Comput. Sci.*, 230(1-2):167–193, 2000.

10 Alexander Drewitz and Alejandro F. Ramiréz. Selected topics in random walk in random environment. *Topics in Percolative and Disordered Systems, Springer Proceedings in Mathematics and Statistics*, 69:23–83, 2014.

11 Ehsan Emamjomeh-Zadeh, David Kempe, and Vikrant Singhal. Deterministic and probabilistic binary search in graphs. In *STOC*, pages 519–532, 2016. [doi:10.1145/2897518.2897656,].

12 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, 1994. [doi:10.1137/S0097539791195877,].

13 Irene Finocchi, Fabrizio Grandoni, and Giuseppe F. Italiano. Resilient search trees. In *SODA*, pages 547–553, 2007. URL: `http://dl.acm.org/citation.cfm?id=1283383.1283442`.

14 Irene Finocchi and Giuseppe F. Italiano. Sorting and searching in the presence of memory faults (without redundancy). In *STOC*, pages 101–110, 2004. [doi:10.1145/1007352.1007375,].

15 Ehud Fonio, Yael Heyman, Lucas Boczkowski, Aviram Gelblum, Adrian Kosowski, Amos Korman, and Ofer Feinerman. A locally-blazed ant trail achieves efficient collective navigation despite limited information, eLife 2016;5:e20185, 2016. URL: `https://elifesciences.org/content/5/e20185`.

16 Nicolas Hanusse, David Ilcinkas, Adrian Kosowski, and Nicolas Nisse. Locating a target with an agent guided by unreliable local advice: How to beat the random walk when you have a clock? In *PODC*, pages 355–364, 2010. [doi:10.1145/1835698.1835781,].

17 Nicolas Hanusse, Dimitris Kavvadias, Evangelos Kranakis, and Danny Krizanc. Memoryless search algorithms in a network with faulty advice. *Theoretical Computer Science*, 402(2–3):190–198, 2008. [doi:10.1016/j.tcs.2008.04.034,].

18 Nicolas Hanusse, Evangelos Kranakis, and Danny Krizanc. Searching with mobile agents in networks with liars. *Discrete Applied Mathematics*, 137(1):69–85, 2004. [doi:10.1016/S0166-218X(03)00189-6,].

19 Richard M. Karp and Robert Kleinberg. Noisy binary search and its applications. In *SODA*, pages 881–890, 2007. URL: `http://dl.acm.org/citation.cfm?id=1283383.1283478`.

20 Eduardo Sany Laber and Loana Tito Nogueira. Fast searching in trees. In *Eletronic Notes on Discrete Mathematics*, 2001.

21 Shay Mozes, Krzysztof Onak, and Oren Weimann. Finding an optimal tree searching strategy in linear time. In *SODA*, pages 1096–1105, 2008. URL: `http://dl.acm.org/citation.cfm?id=1347082.1347202`.

22 Krzysztof Onak and Pawel Parys. Generalization of binary search: Searching in trees and forest-like partial orders. In *FOCS*, pages 379–388, 2006. [doi:10.1109/FOCS.2006.32,].

23 Andrzej Pelc. Searching games with errors - fifty years of coping with liars. *Theor. Comput. Sci.*, 270(1-2):71–109, 2002.

24 Alain-Sol Snitzman. Topics in random walks in random environment. *ICTP Lecture Notes Series*, 2004.

Efficient and Adaptive Parameterized Algorithms on Modular Decompositions

Stefan Kratsch
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
kratsch@informatik.hu-berlin.de

Florian Nelles
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
nelles@informatik.hu-berlin.de

Abstract

We study the influence of a graph parameter called modular-width on the time complexity for optimally solving well-known polynomial problems such as MAXIMUM MATCHING, TRIANGLE COUNTING, and MAXIMUM s-t VERTEX-CAPACITATED FLOW. The modular-width of a graph depends on its (unique) modular decomposition tree, and can be computed in linear time $\mathcal{O}(n+m)$ for graphs with n vertices and m edges. Modular decompositions are an important tool for graph algorithms, e.g., for linear-time recognition of certain graph classes.

Throughout, we obtain efficient parameterized algorithms of running times $\mathcal{O}(f(\mathsf{mw})n + m)$, $\mathcal{O}(n + f(\mathsf{mw})m)$, or $\mathcal{O}(f(\mathsf{mw}) + n + m)$ for low polynomial functions f and graphs of modular-width mw. Our algorithm for MAXIMUM MATCHING, running in time $\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw}\, n + m)$, is both faster and simpler than the recent $\mathcal{O}(\mathsf{mw}^4\, n + m)$ time algorithm of Coudert et al. (SODA 2018). For several other problems, e.g., TRIANGLE COUNTING and MAXIMUM b-MATCHING, we give adaptive algorithms, meaning that their running times match the best unparameterized algorithms for worst-case modular-width of $\mathsf{mw} = \Theta(n)$ and they outperform them already for $\mathsf{mw} = o(n)$, until reaching linear time for $\mathsf{mw} = \mathcal{O}(1)$.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases efficient parameterized algorithms, modular-width, adaptive algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.55

Related Version See [26], `https://arxiv.org/abs/1804.10173`, for the full version of the paper.

1 Introduction

Determining the best possible worst-case running times for computational problems lies at the heart of algorithmic research. For many intensively studied problems progress has been stalled for decades and one may suspect that the "correct" running times have already been found. While there is still only little known regarding unconditional lower bounds, the recent success of "fine-grained analysis of algorithms" has brought plenty of tight conditional lower bounds for a wealth of problems (see, e.g., [31, 5, 2]). Indeed, if one is willing to believe in the conjectured worst-case optimality of known algorithms for 3-SUM, ALL-PAIRS-SHORTEST PATHS (APSP), or SATISFIABILITY[1] then lots of other known algorithms must be optimal as well. Even if there is no general agreement on the truth of the conjectures, the previously

[1] It has been conjectured that there is no $\mathcal{O}(n^{2-\varepsilon})$ time algorithm for 3-SUM, no $\mathcal{O}(n^{3-\varepsilon})$ time for APSP, and there is no $c < 2$ such that k-SAT can be solved in time $\mathcal{O}(c^n)$ for each fixed k (SETH).

© Stefan Kratsch and Florian Nelles;
licensed under Creative Commons License CC-BY

stalled work can now be focused on beating the best known times for just those problems rather than for a multitude of problems. Complementary to the quest for refuting conjectures and beating long-standing fastest algorithms, what should we do if the conjectures and implied lower bounds are true (or if we simply fail to disprove them)? Certainly, quadratic or cubic time is often too slow, even long before entering the realm of big data. Apart from heuristics and approximate algorithms, a possible solution lies in taking advantage of structure in the input and deriving worst-case running times that depend on parameters that quantify this structure. Consider for example the LONGEST COMMON SUBSEQUENCE problem, where a breakthrough result [1, 6] proved that there is no $\mathcal{O}(n^{2-\varepsilon})$ time algorithm for any $\varepsilon > 0$ unless SATISFIABILITY can be solved in $\mathcal{O}((2-\varepsilon')^n)$ time for some $\varepsilon' > 0$ and SETH fails. Long before this result, algorithms were discovered that run much faster than $\mathcal{O}(n^2)$ time when certain parameters are small (cf. [7]); curiously, a very recent result of Bringmann and Künnemann [7] shows that these are optimal modulo SETH (while giving one new optimal algorithm for binary alphabets). Similarly, for the task of sorting an array of n items, there is the (unconditional) lower bound of $\Omega(n \log n)$ for comparison-based sorting, which is matched by well-known sorting algorithms. The goal in the area of adaptive sorting is to find algorithms that are adaptive to presortedness (a.k.a., input structure) with very low running times for almost sorted inputs while maintaining competitive running times as disorder increases (cf. [12]).

The success of fine-grained analysis has rekindled the interest in outperforming (possibly optimal) worst-case running times by tailoring algorithms to benefit from input structure. This fits naturally into the framework of parameterized complexity where running times are expressed in terms of input size and one or more problem-specific parameters. Usually, this is aimed at NP-hard problems and a key goal is to obtain *fixed-parameter tractable* (FPT) algorithms that run in time $f(k)n^c$ where $f(k)$ is a (usually exponential) function of the parameter and n^c denotes a fixed polynomial in the input size n. Recent work of Giannopoulou et al. [19] has initiated a programmatic study of what they called "FPT in P", i.e., efficient parameterized algorithms for tractable problems. Here, they propose to seek running time $\mathcal{O}(k^\alpha n^\beta)$ when the best dependence on input size alone is $\mathcal{O}(n^\gamma)$ for $\gamma > \beta$; in particular, algorithms with linear dependence on the input size are sought, i.e., time $\mathcal{O}(k^\alpha n)$. Giannopoulou et al. suggest that MAXIMUM MATCHING could become a focal point of study, similar to the related NP-hard VERTEX COVER problem in parameterized complexity.

There have been several recent publications that fit into the FPT in P program [14, 28, 4, 13, 24]. Several works focus on the *treewidth* parameter, which is of core importance in parameterized complexity [14, 23]. In particular, Fomin et al. [14] obtained algorithms that depend polynomially on input size n and treewidth tw to solve a number of problems related to determinants and systems of linear inequalities; e.g., they can solve MAXIMUM MATCHING in time $\mathcal{O}(\mathsf{tw}^3\, n \log n)$ and vertex flow with unit capacities in time $\mathcal{O}(\mathsf{tw}^2\, n \log n)$. (A small caveat of treewidth in this context is that it is NP-hard to compute so one has to resort to an approximation with polynomial blow-up in the treewidth.) Iwata et al. [24] studied the related parameter *tree-depth* and, among other results, showed how to solve MAXIMUM MATCHING in time $\mathcal{O}(\mathsf{td}\, m)$ on graphs of tree-depth td. Very recently, Coudert et al. [8] studied another tree-width related parameter called *clique-width* as well as several related parameters such as modular-width and split-width; they obtain upper and lower bounds for a variety of problems. Their main result is an algorithm for MAXIMUM MATCHING that runs in $\mathcal{O}(\mathsf{mw}^4\, n + m)$ time, where mw stands for the modular-width of the input graph. Note that modular-width and the modular decomposition of a graph can be computed in linear time $\mathcal{O}(n + m)$; the modular-width is an upper bound for the (NP-hard) clique-width but it is itself unbounded already on graphs of constant clique-width.

Table 1 Overview about our results. We denote with n and m the number of vertices and edges, mw denotes the modular-width of the input graph, and λ denotes the edge-connectivity of the graph (which is upper-bounded by the minimum degree δ, so $\lambda \leq \delta \leq 2m/n$. The previous best result for MAXIMUM MATCHING, parameterized by modular-width mw, was $\mathcal{O}(\mathsf{mw}^4\, n + m)$ [8].

Problem	Best unparameterized	Our result
MAXIMUM MATCHING	$\mathcal{O}(m\sqrt{n})$ [29]	$\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw}\, n + m)$
MAXIMUM b-MATCHING[2]	$\mathcal{O}((n \log n) \cdot (m + n \log n))$ [16]	$\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw}\, n + m)$ or $\mathcal{O}((\mathsf{mw} \log \mathsf{mw}) \cdot (m + n \log \mathsf{mw}))$
TRIANGLE COUNTING	$\mathcal{O}(n^{\omega})$ [32] or $\mathcal{O}(m^{\frac{2\omega}{\omega+1}}) = \mathcal{O}(m^{1.41})$ [3]	$\mathcal{O}(\mathsf{mw}^{\omega-1}\, n + m)$
EDGE-DISJOINT s-t PATHS	$\mathcal{O}(n^{\frac{3}{2}} m^{\frac{1}{2}})$ [20]	$\mathcal{O}(\mathsf{mw}^3 + n + m)$
GLOBAL MIN CUT	$\mathcal{O}(m + \lambda^2 n \log(n/\lambda))$ [15]	$\mathcal{O}(\mathsf{mw}^3 + n + m)$
MAX s-t VERTEX FLOW	$\mathcal{O}(nm)$ [30]	$\mathcal{O}(\mathsf{mw}^3 + n + m)$
GLOBAL VERTEX MIN CUT	$\mathcal{O}(n^3 \log n)$ [22]	$\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw}\, n + m)$

Our work. We further explore the algorithmic applications of modular-width for well-studied tractable problems. See Table 1 for an overview of our results. First, we improve the running time for MAXIMUM MATCHING from $\mathcal{O}(\mathsf{mw}^4\, n + m)$ to $\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw}\, n + m)$. We follow the same natural recursive approach as in previous work, i.e., computing optimal solutions in a bottom-up fashion on the modular decomposition tree. Unlike Coudert et al. [8], however, we do not seek to use the structure of modules to speed up the computation of augmenting paths, starting from an union of maximum matchings for the child modules. Instead, we simplify the current graph, while retaining the same maximum matching size, such that the found solutions can be encoded into vertex capacities in a graph with at most $3\,\mathsf{mw}$ vertices. This allows us to forget the matchings for the modules and instead of augmenting paths it suffices to find a maximum b-matching subject to vertex capacities; using an $\mathcal{O}(\min\{b(V), n \log n\} \cdot (m + n \log n)) = \mathcal{O}(n^3 \log n)$ time algorithm due to Gabow [16] then yields the claimed running time.[3]

Our algorithm for MAXIMUM MATCHING easily generalizes to computing maximum b-matchings in the same time $\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw}\, n + m)$. By a different summation of the running time, one can also bound the time by $\mathcal{O}((\mathsf{mw} \log \mathsf{mw}) \cdot (m + n \log \mathsf{mw}))$. For large total capacity $b(V)$, Gabow's algorithm runs in time $\mathcal{O}((n \log n) \cdot (m + n \log n))$, which matches our running time for graphs with worst-case modular-width of $\mathsf{mw} = \Theta(n)$.

Thus, when capacities are large, our algorithm interpolates smoothly between linear time $\mathcal{O}(n + m)$ for $\mathsf{mw} = \mathcal{O}(1)$ and the running time of the best unparameterized algorithm for $\mathsf{mw} = \Theta(n)$; i.e., it is an *adaptive algorithm* and already $\mathsf{mw} = o(n)$ gives an improved running time. Such adaptive algorithms (for other problems and parameter) were also considered by Iwata et al. [24]. For MAXIMUM MATCHING, the comparison with the $\mathcal{O}(m\sqrt{n})$ time algorithm of Micali and Vazirani [29] is of course less favorable, but still yields a fairly large regime for mw where we get a faster algorithm.

We next study TRIANGLE COUNTING where, given a graph $G = (V, E)$, we need to determine the number of triangles in G. The fastest known algorithm in terms of n relies on fast matrix multiplication and runs in $\mathcal{O}(n^{\omega})$ time [32] where ω is the matrix multiplication

[2] For $b(V) \geq n \log n$

[3] The obvious upper bound of $\mathcal{O}(\mathsf{mw}^3 \log \mathsf{mw}\, n + m)$ of applying Gabow's algorithm on each prime node can be improved by a slightly more careful summation; the same applies in the other results.

exponent.[4] We present an algorithm that runs in $\mathcal{O}(\mathsf{mw}^{\omega-1}\, n+m)$ time. Again, our running time smoothly interpolates between linear time $\mathcal{O}(n+m)$ for $\mathsf{mw} = \mathcal{O}(1)$ and the best unparameterized time for $\mathsf{mw} = \Theta(n)$, making it adaptive for sufficiently dense graphs; else, the $\mathcal{O}(m^{\frac{2\omega}{\omega+1}}) = \mathcal{O}(m^{1.41})$ time algorithm of Alon et al. [3] is faster. Coudert et al. [8] obtained time $\mathcal{O}(\mathsf{cw}^2(n+m))$ where cw is the clique-width of G; this is incomparable with our result because clique-width is a smaller parameter ($\mathsf{cw} \leq \mathsf{mw}$ and there are graphs with $\mathsf{cw} = \mathcal{O}(1)$ but $\mathsf{mw} = \Theta(n)$) but (so far) allows a worse time.

Finally, we turn to problems related to edge- and vertex-disjoint paths. Due to space restrictions the discussion of those problems are deferred to the full version [26]. Our results for the vertex-disjoint paths generalize to vertex-capacitated flows and global min cuts; it is easy to see that there is little use for modular-width for most edge-weighted/capacitated problems because it suffices to solve them on cliques, which have modular-width equal to two (see also Section 5). Note that standard transformations between different variants of path- and flow-type problems do not apply here because they affect the modular-width of the graph. We obtain the following running times: MAXIMUM s-t VERTEX-CAPACITATED FLOW in $\mathcal{O}(\mathsf{mw}^3 + n + m)$ time; GLOBAL VERTEX-CAPACITATED MIN CUT in $\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw}\, n + m)$ time; EDGE-DISJOINT s-t PATHS in $\mathcal{O}(\mathsf{mw}^3 + n + m)$ time; and UNWEIGHTED GLOBAL MIN CUT in $\mathcal{O}(\mathsf{mw}^3 + n + m)$ time. The running times for flows/paths are linear in the graph size and only have an additive contribution in terms of the modular-width, because at most one involved computation (on a prime node) is needed. These also give rise to linear-time kernelization-like algorithms that return an equivalent instance of size $poly(\mathsf{mw})$, which is the one instance that one would run some other algorithm on (i.e., the only source of non-linear time). Such results (for other problems) have also been observed by Coudert et al. [8]. It is easy to see that *any* algorithm of running time $\mathcal{O}(f(k) + n + m)$, for some parameter k, implies a linear-time kernelization: Run the algorithm for $c(n+m)$ steps, for sufficiently large c relative to hidden constants in $\mathcal{O}$; it either terminates and returns the correct answer or allows the conclusion that $n + m < f(k)$, i.e., the input instance itself is the kernel. Again, as done for MAXIMUM b-MATCHING, one can obtain different bounds for the running time by slightly different summations. For example, the running time for MAXIMUM s-t VERTEX-CAPACITATED FLOW can also be bounded by $\mathcal{O}(\mathsf{mw}\, m + n)$, meaning that the algorithm is never worse than the optimal unparameterized algorithm and outperforms it already for $\mathsf{mw} = o(n)$.

To summarize, we obtain several results that fit into the recent FPT in P program (and the much older programs of adaptive algorithms and faster algorithms for restricted settings), i.e., efficient parameterized algorithms with running times $\mathcal{O}(poly(\mathsf{mw})(n+m))$ or $\mathcal{O}(poly(\mathsf{mw}) + m + n)$. All running times are linear for $\mathsf{mw} = \mathcal{O}(1)$ and several algorithms are adaptive so that they match the best known algorithm for $\mathsf{mw} = \Theta(n)$ and outperform it already when $\mathsf{mw} = o(n)$, possibly only for sufficiently dense graphs. Of course, we use the best algorithms as black boxes so the message is that throughout there is little to no overhead even in the worst case for using a modular decomposition-based approach and getting savings in running time already for large (but not worst-case) modular-width.

Related work. TRIANGLE COUNTING is solvable in time $\mathcal{O}(n^{\omega})$ using fast matrix multiplication [3], and even for the simpler TRIANGLE DETECTION problem, where only (non-)existence of a single triangle needs to be reported, it has been conjectured that there is no $\mathcal{O}(n^{\omega-\varepsilon})$

[4] It is known that $2 \leq \omega < 2.3728639$ due to Le Gall [17]. By definition of ω the running time is in fact $\mathcal{O}(n^{\omega+o(1)})$; adopting a common abuse of notation we use exponent ω for brevity.

time and no combinatorial $\mathcal{O}(n^{3-\varepsilon})$ time algorithm. The fastest known algorithm for counting triangles in sparse graphs is the AYZ algorithm due to Alon, Yuster, and Zwick [3], which runs in time $\mathcal{O}(m^{\frac{2\omega}{\omega+1}})$ ($\mathcal{O}(m^{1.41})$ for $\omega < 2.373$). Coudert et al. [8] gave a faster algorithm for graphs of bounded clique-width cw, running in time $\mathcal{O}(\mathsf{cw}^2(n+m))$. Bentert et al. [4] have studied TRIANGLE ENUMERATION under various parameters including feedback edge number, distance to d-degenerate graphs, and clique-width. The latter one outputs all triangles in time $\mathcal{O}(\mathsf{cw}^2\, n + n^2 + \#T)$ where $\#T$ denotes the number of triangles in G.

The currently best maximum flow algorithm is due to Orlin [30] and runs in time $\mathcal{O}(nm)$. Using a flow algorithm, one can determine the number of edge- or vertex-disjoint s-t paths in a graph, but in the unweighted case one can do slightly better, e.g., computing the number of edge-disjoint paths in an undirected graph can be done in time $\mathcal{O}(n^{\frac{3}{2}}m^{\frac{1}{2}})$ using an algorithm due to Goldberg and Rao [20]. Finding a global minimum edge cut with weights on the edges in an undirected graph can be done in time $\mathcal{O}(nm + n^2 \log n)$ due to Stoer and Wagner [33]. The unweighted variant can be solved in time $\mathcal{O}(m + \lambda^2 n \log(n/\lambda))$ by Gabow [15], where λ denotes the edge-connectivity of the graph (which is upper-bounded by the minimum degree δ, so $\lambda \leq \delta \leq 2m/n$). There is also a randomized algorithm with running time $\mathcal{O}(m \log^3 n)$ due to Karger [25].

The notion of a modular decomposition was first introduced by Gallai [18] for recognizing comparability graphs. The first linear time algorithm to compute a modular decomposition was independently developed by McConnell and Spinrad [27] and Cournier and Habib [9]. Tedder et al. [34] later gave a new and much simpler linear-time algorithm.

Organization. In Section 2 we briefly introduce basic notation, define the modular decomposition tree, and define modular-width. Then, in Section 3, we consider the problem MAXIMUM MATCHING and the generalization to MAXIMUM b-MATCHING. In Section 4, we study the problem TRIANGLE COUNTING. Due to space restrictions, the remaining results for edge/vertex-disjoint paths, flows, and cuts can be found in the full version [26]. We conclude in Section 5.

2 Preliminaries

We use standard graph notation [10]. An s-t *vertex-capacitated flow* in a graph $G = (V, E)$ with vertex capacities $c\colon V \to \mathbb{R}$ is a weighted collection of s-t paths in G such that the total weight of paths including any vertex $v \in V \setminus \{s,t\}$ is at most the capacity $c(v)$. (Equivalently, one may define this as a function $f\colon E(\overleftrightarrow{G}) \to \mathbb{R}$ where $\overleftrightarrow{G} = (V, A)$ with $A = \{(u,v),(v,u) \mid \{u,v\} \in E\}$ that has flow-conservation at each $v \in V \setminus \{s,t\}$ and with $\sum_{(u,v)\in\delta^-_{\overleftrightarrow{G}}(v)} f((u,v)) \leq c(v)$ for all $v \in V \setminus \{s,t\}$, where $\delta^-_{\overleftrightarrow{G}}(v)$ is the set of arcs with end in v.) The value of such a flow, denoted by $|f|$, is the total weight over all the s-t paths (equivalently, $\sum_{(v,t)\in\delta^-_{\overleftrightarrow{G}}(t)} f(v,t)$). For unit capacities $c \equiv 1$ this is equivalent to a maximum collection of vertex-disjoint s-t paths.

We say that two sets A and B *overlap* if $A \cap B \neq \emptyset$, $A \setminus B \neq \emptyset$, and $B \setminus A \neq \emptyset$ and let $[n] = \{1, 2, \ldots, n\}$ for any $n \in \mathbb{N}$.

Modular Decomposition. Let $G = (V, E)$ be a graph. A *module* is a vertex set $M \subseteq V$ such that all vertices in M have the same neighborhood in $V \setminus M$. In other words, $M \subseteq N(x)$ or $M \cap N(x) = \emptyset$ for every vertex $x \in V \setminus M$. Clearly, $\emptyset$, V, and $\{v\}$ for every $v \in V$ are modules of G; these are called *trivial modules*. If a graph only admits trivial modules, we call G *prime*. Consider a partition $P = \{M_1, M_2, \ldots, M_\ell\}$ of the vertices of G into modules where $\ell \geq 2$, called *modular partition*. If there is $v \in M_i$ and $u \in M_j$ with $\{u,v\} \in E$, then any vertex in M_i is adjacent to every vertex in M_j. In this case, we can call two modules M_i and M_j of P *adjacent*, and *non-adjacent* otherwise.

▶ **Definition 1.** Let $P = \{M_1, M_2, \dots, M_\ell\}$ be a modular partition of a graph $G = (V, E)$. The *quotient graph* $G_{/P} = (\{q_{M_1}, q_{M_2}, \dots, q_{M_\ell}\}, E_P)$ is the graph whose vertices are in a one-to-one correspondence to the modules in P. Two vertices q_{M_i}, q_{M_j} of $G_{/P}$ are adjacent if and only if the corresponding modules M_i and M_j are adjacent (with adjacency as above).

If $P = \{M_1, M_2, \dots, M_\ell\}$ is a modular partition of a graph G, then the quotient graph $G_{/P}$ is a compact representation of the edges with endpoint in different modules. Together with all subgraphs $G[M_i]$, with $i \in [\ell]$, we can reconstruct G. Each subgraph $G[M_i]$ is called a *factor*. Instead of specifying the factors, one can recursively decompose them as well until one reaches trivial modules $\{v\}$. To make the decomposition unique, one considers modular partitions consisting of strong modules. A module of a graph G is called a *strong* module, if it does not overlap with any other module of G. One can represent all strong modules of a graph G by an inclusion tree $MD(G)$. Each strong module M in G corresponds to a vertex v_M in $MD(G)$. A vertex v_A is an an ancestor of v_B in $MD(G)$ if and only if $B \subsetneq A$ for the corresponding strong modules A and B of G. Hence, the root node of $MD(G)$ corresponds always to the complete vertex set V of G and every leaf of $MD(G)$ corresponds a singleton set $\{v\}$ with $v \in V$. Consider an internal node v_M of $MD(G)$ with the set of children $\{v_{M_1}, \dots, v_{M_\ell}\}$, i.e., v_M corresponds to a strong module M of G and $P = \{M_1, \dots, M_\ell\}$ is a modular partition of $G[M]$ into strong modules where M_i is the corresponding module of v_{M_i}, with $i \in [\ell]$. There are three types of internal nodes in $MD(G)$. A node v_M in $MD(G)$ is *degenerate*, if for any non-empty subset of the children of v_M in $MD(G)$, the union of the corresponding modules induces a (not necessarily strong) module. In this case the quotient graph $G[M]_{/P}$ is either a clique or an independent set. In the former case one calls v_M a *series* node, in the later a *parallel* node. Another case are so called *prime* nodes. Here, for no proper subset of the children of v_M, the union of the corresponding modules induces a module. In this case the quotient graph of v_M is prime. Gallai showed there are no further nodes in $MD(G)$.

▶ **Theorem 2** ([18]). *For any graph $G = (V, E)$ one of the three conditions is satisfied:*
- *G is not connected,*
- *$\overline{G}$ is not connected,*
- *G and $\overline{G}$ are connected and the quotient graph $G_{/P}$, where P is the maximal modular partition of G, is a prime graph.*

Theorem 2 implies that $MD(G)$ is unique. The tree $MD(G)$ is called the *modular decomposition tree* and the *modular-width*, denoted by $\mathsf{mw} = \mathsf{mw}(G)$, is the minimum $k \geq 2$ such that any prime node in $MD(G)$ has at most k children. Since every node in $MD(G)$ has at least two children and there are exactly n leaves, $MD(G)$ has at most $2n - 1$ nodes. It is known that $MD(G)$ can be computed in time $\mathcal{O}(n + m)$ [34]. We refer to a survey of Habib and Paul [21] for more information.

3 Maximum Matching

In the MAXIMUM MATCHING problem we are given a graph $G = (V, E)$ and need to find a maximum set $X \subseteq E$ of pairwise disjoint edges. The size of a maximum matching of a graph G is denoted by $\mu(G)$. Edmond [11] was the first to give a polynomial-time algorithm for this problem. The fastest known algorithm, due to Micali and Vazirani [29], runs in time $\mathcal{O}(m\sqrt{n})$. A *b-matching* is a generalization of a matching that specifies for each vertex a *degree bound* of how many edges in the matching may be incident with that vertex. Formally, degree bounds are given by a function $b\colon V \to \mathbb{N}$, and a b-matching is a function $x\colon E \to \mathbb{N}$ that fulfills for every vertex $v \in V$ the constraint that $\sum_{e\in\delta(v)} x(e) \leq b(v)$. Gabow [16] showed how to find a b-matching that maximizes $\sum_{e\in E} x(e)$ in time $\mathcal{O}((n \log n) \cdot (m + n \log n))$.

Recently, Coudert et al. [8] gave an $\mathcal{O}(\mathsf{mw}^4\, n+m)$ time algorithm for MAXIMUM MATCHING, where mw denotes the modular-width of the input graph. In the following we will improve this result by providing an algorithm for MAXIMUM MATCHING that runs in time $\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw} \cdot n + m)$. The main idea of our algorithm is to compress the computation of a matching in G to a computation of a b-matching, instead of using the structure of modular decompositions to speed up the search for augmenting paths (like in [8]).

▶ **Theorem 3.** *For every graph $G = (V, E)$ with modular-width mw, MAXIMUM MATCHING can be solved in time $\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw} \cdot n + m)$.*

Algorithm. First, we compute the modular decomposition tree $MD(G)$. We will traverse the decomposition tree in a bottom-up manner. For each v_M in $MD(G)$, with M denoting the corresponding module of G, we will compute a maximum matching in $G[M]$. Note that for the root module v_M of $MD(G)$ it holds that $G[M] = G$. For any leaf module v_M of $MD(G)$, we have $\mu(G[M]) = 0$, since $G[M]$ is a graph consisting of a single vertex. Let v_M be a non-leaf vertex in $MD(G)$ with the set of children $\{v_{M_1}, \dots, v_{M_\ell}\}$. This means that $\{M_1, \dots, M_\ell\}$ is a modular partition of $G[M]$, where $M_i \subseteq M$ corresponds to the vertex v_{M_i} in $MD(G)$ for $i \in [\ell]$. In the following, we can always assume that we have already computed $\mu(G[M_i])$ for $i \in [\ell]$. The next lemma shows that the concrete structure inside a module is irrelevant for the maximum matching size of the whole graph, i.e., only the number of vertices and the maximum matching size is important. The lemma is a more general version of [8, Lemma 5.1], but can be proven in a similar way.

▶ **Lemma 4.** *Let M be a module of $G = (V, E)$ and let $G[M] = (M, E_M)$. Let $A \subseteq \binom{M}{2}$ be any set of edges on the vertices of M such that $\mu((M, A)) = \mu((M, E_M))$. Then, the size of a maximum matching of $G' = (V, (E \setminus E_M) \cup A)$ is equal to the size of a maximum matching of G.*

Proof. We first show that $\mu(G') \geq \mu(G)$. Let us consider a maximum matching $F \subseteq E$ in $G = (V, E)$. To get a maximum matching in G' we replace all edges in F that are incident with M: First, replace all edges in $F \cap E(G[M])$ by an arbitrary matching $A' \subseteq A$ of the same size; such a matching must exist because $F \cap E(G[M])$ is not larger than a maximum matching in $G[M]$ and $\mu((M, A)) = \mu((M, E_M))$. Second, we replace all edges in F that have exactly one endpoint in M as follows: Let $X \subseteq M$ be the set of vertices in M that are endpoints of an edge in F whose other endpoint is not in M. By assumption, $|M \setminus V(A')| \geq |X|$ and since all vertices in $V \setminus M$ that are connected to a vertex in X in G are also connected to all vertices in $M \setminus V(A')$ in G', we can replace all edges of F that have exactly one endpoint in M. Thus, $\mu(G') \geq \mu(G)$, i.e., replacing the edges in a module by an arbitrary set of edges with same maximum matching size does not decrease the size of the maximum matching for the whole graph. Applying this argument for $A' := E_M$ to swap back to the original edge set yields, $\mu(G) \geq \mu(G')$ and completes the proof. ◀

We now describe how to compute $\mu(G[M])$ for a node v_M in $MD(G)$. Let $\{v_{M_1}, \dots, v_{M_\ell}\}$ be the set of children of v_M in $MD(G)$, meaning that $P = \{M_1, \dots, M_\ell\}$ is a modular partition of $G[M]$. We can assume that we have already computed $\mu(G[M_i])$ for $i \in [\ell]$.

Let $G[M]_{/P}$ be the quotient graph of $G[M]$. If v_M is a parallel node then $G[M]_{/P}$ is edgeless, i.e., $G[M]$ is the disjoint union of all $G[M_i]$. In this case a maximum matching for $G[M]$ simply consists of the union of maximum matchings for each $G[M_i]$ and we set $\mu(G(M)) = \sum_{i\in[\ell]} \mu(G[M_i])$. Next, suppose that v_M is a prime node. We will reduce the problem of computing a maximum matching in $G[M]$ to computing a maximum b-matching in an auxiliary graph closely related to the quotient graph of v_M that we will define next.

▶ **Definition 5.** Let $G = (V, E)$ be a graph and $P = \{M_1, \ldots, M_\ell\}$ be a modular partition of G. Let n_i denote the number of vertices in $G[M_i]$ and f_i the size of a maximum matching in $G[M_i]$. We define an auxiliary graph $G^* = (V^*, E^*)$ together with degree bounds $b^* : V^* \to \mathbb{N}$ as an instance (G^*, b^*) for the maximum b-matching problem as follows:

- For every module $M_i \in P$, with $i \in [\ell]$, we add three vertices v_i^1, v_i^2, v_i^3 to V^* and set $b^*(v_i^1) = b^*(v_i^2) = f_i$ and $b^*(v_i^3) = n_i - 2f_i$.
- We add the edge $\{v_i^1, v_i^2\}$ to E^* for $i \in [\ell]$.
- For each edge between vertices q_i and q_j in $G_{/P}$ that corresponds to modules M_i and M_j, we add the nine edges $\{v_i^c, v_j^d\}$ with $c, d \in \{1, 2, 3\}$ to E^*.

▶ **Lemma 6.** *Let $G = (V, E)$ be a graph and $P = \{M_1, \ldots, M_\ell\}$ be a modular partition of G. Let (G^*, b^*) be the instance of a maximum b-matching problem as defined in Definition 5. Then the size of maximum matchings in G is equal to the size of a maximum b-matching of (G^*, b^*).*

Proof. Consider a graph $G = (V, E)$ with a modular partition $P = \{M_1, \ldots, M_\ell\}$. For $M_i \in P$ let $n_i = |V(G[M_i])|$ and let $f_i = \mu(G[M_i])$. Due to Lemma 4, we can replace each $G[M_i]$, for $i \in [\ell]$, by a graph consisting of a complete bipartite graph K_{f_i,f_i} together with $n_i - 2f_i$ single vertices without changing the size of a maximum matching. We do this for every module $M_i \in P$ and denote the resulting graph by $\overline{G}$. Note, that $\mu(G) = \mu(\overline{G})$. Now, each replacement of $G[M_i]$ can be partitioned into three modules, namely the two parts of the complete bipartite graph K_{f_i,f_i} and the one set consisting of $n_i - 2f_i$ single vertices. This results in a modular partition P' of $\overline{G}$ of size 3ℓ, and for every module $M \in P'$ the factor graph $G[M]$ is an independent set. The quotient graph $\overline{G}_{/P'}$ is exactly the auxiliary graph G^* of G and the degree bound of a vertex v in G^* is equal to the number of vertices in the corresponding module. Since solving a b-matching in (G^*, b^*) directly corresponds to solving maximum matching in $\overline{G}$, this completes the proof. ◀

Finally, suppose that v_M is a series node. Instead of computing $\mu(G[M])$ directly, we will modify the decomposition tree $MD(G)$ (cf. [8]). Let $\{v_{M_1}, \ldots, v_{M_\ell}\}$ be the children of v_M in $MD(G)$. We will iteratively compute a maximum matching for $G_i = G[\cup_{1\le j\le i} M_j]$ by using a modular partition of G_i consisting of the two modules $\cup_{1\le j<i} M_j$ and M_i, for $i \in [\ell]$. This means that we replace a series node with ℓ children by $\ell - 1$ series nodes with only two children. We will treat the newly inserted nodes as prime nodes (with a quotient graph isomorphic to K_2). After replacing the series nodes of the modular decomposition tree $MD(G)$, every node still has at least two children; hence, we still have a most $2n - 1$ nodes in $MD(G)$.

Running Time. Consider a graph $G = (V, E)$ with modular-width mw. Computing the modular decomposition tree $MD(G)$ takes time $\mathcal{O}(n + m)$. Since there are at most $2n - 1$ nodes in $MD(G)$ the total computation for all parallel nodes together takes time $\mathcal{O}(n)$. As described above, we modify the decomposition tree such that every series node of $MD(G)$

with $\ell \geq 3$ children is replaced by $\ell - 1$ 'pseudo-prime' nodes with exactly two children. This replacement can be done in time $O(n)$. Now, every node $v_M \in MD(G)$ that is not a parallel node has a set of children $\{v_{M_1}, \dots, v_{M_\ell}\}$ with $\ell \leq \mathsf{mw}$. This means that $P = \{M_1, \dots, M_\ell\}$ is a modular partition of $G[M]$ and the quotient graph $G[M]_{/P}$ consists of $\ell \leq \mathsf{mw}$ vertices. Since we have already computed $\mu(G[M_i])$ for all $i \in [\ell]$, we can construct the auxiliary graph G^* of $G[M]$ as defined in Definition 5 in time $\mathcal{O}(V(G^*) + E(G^*)) = \mathcal{O}(\ell^2)$. Recall, that $|V(G^*)| = 3\ell$. Thus, we can compute a maximum b-matching of G^* subject to b in time $\mathcal{O}(\ell^3 \log \ell)$ using the algorithm due to Gabow [16]. We have to do this for every prime and series node, but a slightly more careful summation of running times over all nodes gives an improvement over the obvious upper bound of $\mathcal{O}(\mathsf{mw}^3 \log \mathsf{mw} \cdot n + m)$: Let t be the number of nodes in $MD(G)$ and for a node v_{M_i} in $MD(G)$ let ℓ_i denote the number of children, i.e. the number of vertices of the quotient graph of $G[M_i]$. Then, neglecting constant factors and assuming that $MD(G)$ is already computed, we can solve MAXIMUM MATCHING, in time:

$$\sum_{i=1}^{t} \ell_i^3 \log \ell_i \leq \left(\sum_{i=1}^{t} \ell_i \right) \cdot \max_{i \in [t]} \{\ell_i^2 \log \ell_i\} \leq 2n \cdot \max_{i \in [t]} \{\ell_i^2 \log \ell_i\} \leq 2n \cdot (\mathsf{mw}^2 \log \mathsf{mw})$$

The second inequality holds, since $\sum_{i=1}^{t} \ell_i$ counts each node in $MD(G)$ once, except for the root. Since constant factors propagate through the inequality, the total running time of the algorithm is $\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw} \cdot n + m)$, which proves Theorem 3.

Generalization to b-matching. We can easily generalize this result to the more general maximum b-matching problem.

▶ **Theorem 7.** *For every graph $G = (V, E)$ with modular-width* mw*,* MAXIMUM b-MATCHING *can be solved in time $\mathcal{O}(\mathsf{mw}^2 \log \mathsf{mw} \cdot n + m)$.*

Again, the concrete structure inside a module will not be important. The only important information is the size of a maximum b-matching and the sum of all b-values in a module. We naturally extend Definition 5 to b-matchings:

▶ **Definition 8.** Let $G = (V, E)$ be a graph with $b \colon V \to \mathbb{N}$ and let $P = \{M_1, \dots, M_\ell\}$ be a modular partition of G. Let $n_i = \sum_{v \in M_i} b(v)$ and f_i be the size of a maximum b-matching in $G[M_i]$ for $i \in [\ell]$. We define the auxiliary graph $G^* = (V^*, E^*)$ together with degree bounds $b^* \colon V \to \mathbb{N}$ in the same way as done in Definition 5.

▶ **Lemma 9.** *Let $G = (V, E)$ be a graph and $P = \{M_1, \dots, M_\ell\}$ be a modular partition of G. Let (G^*, b^*) be the instance of a maximum b-matching problem as defined in Definition 8. Then the size of a maximum b-matching in (G, b) is equal to the size of a maximum b-matching of (G^*, b^*).*

Proof. Consider a graph $G = (V, E)$ with a modular partition $P = \{M_1, \dots, M_\ell\}$. For $M_i \in P$ let $n_i = \sum_{v \in M_i} b(v)$ and let f_i be the size of a maximum b-matching in M_i. Note, that one can solve b-matching by replacing every vertex v by $b(v)$ copies that are connected in the same way as v. After considering this replacement and due to Lemma 4, we can replace $G[M_i]$, for $i \in [\ell]$, by a graph consisting of a complete bipartite graph K_{f_i, f_i} together with $n_i - 2f_i$ single vertices without changing the size of a maximum matching. We do this for every module M_i and denote the resulting graph by $\overline{G}$. As in the proof of Lemma 6, we can subdivide every module in three parts. This yields to the instance (G^*, b^*) as defined in Definition 8. Again, solving a maximum b-matching of (G^*, b^*) directly corresponds to solving a maximum b-matching in $\overline{G}$, which completes the proof. ◀

The running time can be bounded in the same way as before. However, to see that this algorithm is also adaptive for sparse graphs (at least for large b-values), we can modify the computation of the running time: Let t be the number of nodes in $MD(G)$. For a node v_{M_i} in $MD(G)$ let n_i denote the number of vertices and m_i denote the number of edges of the quotient graph of $G[M_i]$. Thus, we can compute a maximum b-matching of G^* subject to b^* in time $\mathcal{O}((n_i \log n_i) \cdot (m_i + n_i \log n_i))$ using the algorithm due to Gabow [16]. Then, neglecting constant factors and assuming that $MD(G)$ is already computed, we can solve MAXIMUM b-MATCHING in time:

$$\begin{aligned}\sum_{i=1}^{t}(n_i \log n_i) \cdot (m_i + n_i \log n_i) &= \sum_{i=1}^{t} m_i n_i \log n_i + \sum_{i=1}^{t} n_i^2 \log^2 n_i \\ &\le \left(\sum_{i=1}^{t} m_i\right) \max_{i\in[t]}\{n_i \log n_i\} + \left(\sum_{i=1}^{t} n_i\right) \max_{i\in[t]}\{n_i \log^2 n_i\} \\ &\le m \cdot \mathsf{mw} \log \mathsf{mw} + 2n \cdot (\mathsf{mw} \log^2 \mathsf{mw})\end{aligned}$$

Since constant factors propagate through the inequality, the total running time of the algorithm is $\mathcal{O}((m + n \log \mathsf{mw}) \cdot (\mathsf{mw} \log \mathsf{mw}))$. Therefore, even for $\mathsf{mw} = \Theta(n)$ our algorithm is not worse than the (currently) best unparameterized algorithm, assuming $b(V) \ge n \log n$, where $b(V) = \sum_{v\in V} b(v)$.

4 Triangle Counting

In this section we consider the TRIANGLE COUNTING problem, in which one is interested in the number of triangles in the input graph.

▶ **Theorem 10.** *For every graph $G = (V, E)$ with modular-width* mw*,* TRIANGLE COUNTING *can be solved in time $\mathcal{O}(n \cdot \mathsf{mw}^{\omega-1} + m)$.*

Algorithm. First, we compute the modular decomposition tree $MD(G)$. We will process $MD(G)$ in a bottom-up manner. For each v_M in $MD(G)$, with corresponding module M in G, we will compute the following three values: the number of vertices $n_M = |V(G[M])|$, the number of edges $m_M = |E(G[M])|$, and the number of triangles t_M in $G[M]$. For any leaf node v_M in $MD(G)$ we have $n_M = 1$ and $m_M = t_M = 0$, because $G[M]$ consists of a single vertex. Let v_M be a non-leaf node in $MD(G)$ with children $\{v_{M_1}, \ldots, v_{M_\ell}\}$. Since we process $MD(G)$ in a bottom-up manner, the values for $G[M_i]$ are already computed for $i \in [\ell]$. If v_M is a parallel node, the values simply add up, i.e. $n_M = \sum_{i=1}^{\ell} n_{M_i}$, $m_M = \sum_{i=1}^{\ell} m_{M_i}$, and $t_M = \sum_{i=1}^{\ell} t_{M_i}$. If v_M is a series node, we will use the same approach as in Section 3 and replace v_M by $\ell - 1$ series nodes with only two children each. Afterwards, we compute the values for a series node v_M with children v_{M_1} and v_{M_2} as follows:

$$\begin{aligned} n_M &= n_{M_1} + n_{M_2} \\ m_M &= m_{M_1} + m_{M_2} + n_{M_1} n_{M_2} \\ t_M &= t_{M_1} + t_{M_2} + m_{M_1} n_{M_2} + m_{M_2} n_{M_1} \end{aligned}$$

Finally, let v_M be a prime node in $MD(G)$ and let $\{v_{M_1}, \ldots, v_{M_\ell}\}$ be the children of v_M in $MD(G)$. This means that $P = \{M_1, \ldots, M_\ell\}$ is a modular partition of $G[M]$. Again, $n_M = \sum_{i=1}^{\ell} n_{M_i}$ and we can compute m_M by traversing all edges in the quotient graph

$G[M]_{/P}$, i.e., $m_M = \sum_{i=1}^{\ell} m_{M_i} + \sum_{\{q_i,q_j\}\in E(G[M]_{/P})} n_{M_i}n_{M_j}$. For computing t_M we count triangles in $G[M]$ of three types: Triangles using vertices in exactly one module, in two (adjacent) modules, or in three modules of P. We call a triangle with vertices in three different modules a *separated* triangle. To compute the number of separated triangles, we use the following lemma:

▶ **Lemma 11.** *Let $G = (V, E)$ be a graph with a modular partition $P = \{M_1, \dots, M_\ell\}$ and quotient graph $G_{/P}$. Let $n_{M_i} := |M_i|$ and consider the weight function $w\colon E(G_{/P}) \to \mathbb{R}^+$ with $w(\{q_i, q_j\}) = \sqrt{n_{M_i}n_{M_j}}$. Let A be the weighted adjacency matrix of $G_{/P}$ with respect to w. Then, the number of separated triangles in G is:*

$$\sum_{i,j=1}^{\ell} \frac{1}{3}(A^2 \circ A)_{i,j},$$

where $A \circ B$ denotes the Hadamard product of the matrices A and B, i.e., $(A \circ B)_{i,j} = A_{i,j}B_{i,j}$.

Proof. To count all separated triangles in G we need to sum up the values $n_{M_i}n_{M_j}n_{M_k}$ for each triangle (q_i, q_j, q_k) in $G_{/P}$. We show, that $(A^2 \circ A)_{i,j}$ exactly corresponds to the number of separated triangles in G with one vertex in M_i and one in M_j; here, a *wedge* is a path on three vertices (and a wedge (q_i, q_k, q_j) requires the presence of edges $\{q_i, q_k\}$ and $\{q_k, q_j\}$):

$$\begin{aligned}
\left(A^2\right)_{i,j} &= \sum_{k=1}^{\ell} A_{i,k}A_{k,j} \\
&= \sum_{\substack{k:(q_i,q_k,q_j) \\ \text{is a wedge in } G_{/P}}} \sqrt{n_{M_i}n_{M_k}}\sqrt{n_{M_k}n_{M_j}} \\
&= \sqrt{n_{M_i}n_{M_j}} \sum_{\substack{k:(q_i,q_k,q_j) \\ \text{is a wedge in } G_{/P}}} n_{M_k} \\
\Rightarrow \left(A^2 \circ A\right)_{i,j} &= \sum_{\substack{k:(q_i,q_k,q_j) \\ \text{is a triangle in } G_{/P}}} n_{M_i}n_{M_j}n_{M_k}
\end{aligned}$$

Since every triangle is counted three times (once for each edge) the lemma follows. ◀

Using Lemma 11, we can compute t_M by

$$t_M = \sum_{i=1}^{\ell} t_{M_i} + \sum_{\{q_i,q_j\}\in E(G_{/P})} \left(m_{M_i}n_{M_j} + n_{M_i}m_{M_j}\right) + \sum_{i,j=1}^{\ell} \frac{1}{3}\left(A^2 \circ A\right)_{i,j},$$

where the three terms refer to triangles with vertices from only one module, triangles using vertices of two adjacent modules, and separated triangles with vertices in three different (pairwise adjacent) modules.

Running Time. Computing the modular decomposition tree $MD(G)$ takes time $\mathcal{O}(n + m)$. Consider a node v_M in $MD(G)$ with children $\{v_{M_1}, \dots, v_{M_\ell}\}$. If v_M is a parallel or a series node then we can compute the values n_M, m_M, and t_M for $G[M]$ in time $\mathcal{O}(\ell)$. Thus, since the number of nodes in $MD(G)$ is at most $2n - 1$, the total running time for all parallel and series nodes is $O(n)$. Assume that v_M is a prime node. Recall, that $P = \{M_1, \dots, M_\ell\}$ is a modular partition of $G[M]$. Computing n_M takes time $\mathcal{O}(\ell)$ and computing m_M takes time $\mathcal{O}(|E(G[M]_{/P})|) = \mathcal{O}(\ell^2)$. The running time for computing t_M is dominated by the computation of A^2, which takes time $\mathcal{O}(\ell^\omega)$. Note, that $2 \leq \ell \leq \mathsf{mw}$. By a similar careful summation as done in Section 3 we can improve the obvious upper bound of $\mathcal{O}(n \cdot \mathsf{mw}^\omega + m)$: Let p be the number of nodes in $MD(G)$ and for v_{M_i} in $MD(G)$ let ℓ_i be the number of children, i.e., the number of vertices of the quotient graph of $G[M_i]$. Neglecting constant factors and assuming that $MD(G)$ is already computed, the running time is:

$$\sum_{i=1}^{p} \ell_i^\omega \leq \left(\sum_{i=1}^{p} \ell_i\right) \max_{i \in [p]} \ell_i^{\omega-1} \leq 2n \cdot \mathsf{mw}^{\omega-1}$$

Again, since constant factors propagate through the inequalities, the total running time of the algorithm is $\mathcal{O}(n \cdot \mathsf{mw}^{\omega-1} + m)$, which proves Theorem 10. Note, that this algorithm is adaptive for dense graphs, meaning that even for $\mathsf{mw} = \Theta(n)$ our algorithm is not worse than $\mathcal{O}(n^\omega)$.

5 Conclusion

We have obtained efficient parameterized algorithms for MAXIMUM MATCHING, MAXIMUM b-MATCHING, TRIANGLE COUNTING, and several path- and flow-type problems with respect to the modular-width mw of the input graph. All time bounds are of form $\mathcal{O}(f(\mathsf{mw})n + m)$, $\mathcal{O}(n + f(\mathsf{mw})m)$, or $\mathcal{O}(f(\mathsf{mw}) + n + m)$, where the latter can be easily seen to imply linear-time preprocessing to size $\mathcal{O}(f(\mathsf{mw}))$. Throughout, the dependence $f(\mathsf{mw})$ is very low and several algorithms are adaptive in the sense that their time bound interpolates smoothly between $\mathcal{O}(n+m)$ when $\mathsf{mw} = \mathcal{O}(1)$ and the best known unparameterized running time when $\mathsf{mw} = \Theta(n)$. Thus, even if typical inputs may have modular width $\Theta(n)$ (a caveat that all structural parameters face to some degree), using these algorithms costs only a constant-factor overhead and already $\mathsf{mw} = o(n)$ yields an improvement over the unparameterized case.

As mentioned in the introduction, (low) modular-width seems useless in problems where edges are associated with weights and/or capacities. Intuitively, these numerical values distinguish edges between adjacent modules M and M', which could otherwise be treated as largely equivalent. For concreteness, consider an instance (G, s, t, w) of the SHORTEST s,t-PATH problem where $w\colon E(G) \to \mathbb{N}$ are the edge weights. Clearly, the distance from s to t is unaffected if we add the missing edges of G and let their weight exceed the sum of weights in w. However, the obtained graph is a clique and has constant modular-width. Similar arguments work for other edge-weighted/capacitated problems like MAXIMUM FLOW using either huge or negligible weights. In each case, running times of form $\mathcal{O}(f(\mathsf{mw})g(n, m))$ would imply time $\mathcal{O}(g(n, m))$ for the unparameterized case (without considering modular-width), so the best such running times cannot be outperformed even for low modular-width.

Apart from developing further efficient (and adaptive?) parameterized algorithms relative to modular-width there are other directions of future work. Akin to conditional lower bounds via fine-grained analysis of algorithms it would be interesting to prove optimality of efficient parameterized algorithms for all regimes of the parameters (e.g., like Bringmann and Künnemann [7]). Which other (graph) parameters allow for adaptive parameterized running times so that even nontrivial upper bounds on the parameter imply faster algorithms than the unparameterized worst case?

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015. doi:10.1109/FOCS.2015.14.

2 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 41–50. ACM, 2015. doi:10.1145/2746539.2746594.

3 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.

4 Matthias Bentert, Till Fluschnik, André Nichterlein, and Rolf Niedermeier. Parameterized aspects of triangle enumeration. In *Fundamentals of Computation Theory - 21st International Symposium, FCT 2017, Bordeaux, France, September 11-13, 2017, Proceedings*, pages 96–110, 2017. doi:10.1007/978-3-662-55751-8_9.

5 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.76.

6 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97, 2015. doi:10.1109/FOCS.2015.15.

7 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1216–1235, 2018. doi:10.1137/1.9781611975031.79.

8 David Coudert, Guillaume Ducoffe, and Alexandru Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2765–2784, 2018. doi:10.1137/1.9781611975031.176.

9 Alain Cournier and Michel Habib. A new linear algorithm for modular decomposition. In Sophie Tison, editor, *Trees in Algebra and Programming - CAAP'94, 19th International Colloquium, Edinburgh, U.K., April 11-13, 1994, Proceedings*, volume 787 of *Lecture Notes in Computer Science*, pages 68–84. Springer, 1994. doi:10.1007/BFb0017474.

10 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

11 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

12 Vladimir Estivill-Castro and Derick Wood. A survey of adaptive sorting algorithms. *ACM Comput. Surv.*, 24(4):441–476, 1992. doi:10.1145/146370.146381.

13 Till Fluschnik, Christian Komusiewicz, George B. Mertzios, André Nichterlein, Rolf Niedermeier, and Nimrod Talmon. When can graph hyperbolicity be computed in linear time? In *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*, pages 397–408, 2017. doi:10.1007/978-3-319-62127-2_34.

14 Fedor V Fomin, Daniel Lokshtanov, Michał Pilipczuk, Saket Saurabh, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on*

Discrete Algorithms, pages 1419–1432. Society for Industrial and Applied Mathematics, 2017.

15 Harold N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *J. Comput. Syst. Sci.*, 50(2):259–273, 1995. doi:10.1006/jcss.1995.1022.

16 Harold N. Gabow. Data structures for weighted matching and extensions to b-matching and f-factors. *CoRR*, abs/1611.07541, 2016. arXiv:1611.07541.

17 François Le Gall. Powers of tensors and fast matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303. ACM, 2014. URL: http://dl.acm.org/citation.cfm?id=2608628, doi:10.1145/2608628.2608664.

18 Tibor Gallai. Transitiv orientierbare graphen. *Acta Mathematica Hungarica*, 18(1-2):25–66, 1967.

19 Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *CoRR*, abs/1506.01652, 2015. arXiv:1506.01652.

20 Andrew V. Goldberg and Satish Rao. Flows in undirected unit capacity networks. *SIAM J. Discrete Math.*, 12(1):1–5, 1999. doi:10.1137/S089548019733103X.

21 Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010. doi:10.1016/j.cosrev.2010.01.001.

22 Jianxiu Hao and James B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *J. Algorithms*, 17(3):424–446, 1994. doi:10.1006/jagm.1994.1043.

23 Thore Husfeldt. Computing graph distances parameterized by treewidth and diameter. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 16:1–16:11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.16.

24 Yoichi Iwata, Tomoaki Ogasawara, and Naoto Ohsaka. On the power of tree-depth for fully polynomial FPT algorithms. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 41:1–41:14, 2018. doi:10.4230/LIPIcs.STACS.2018.41.

25 David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000. doi:10.1145/331605.331608.

26 Stefan Kratsch and Florian Nelles. Efficient and adaptive parameterized algorithms on modular decompositions. *CoRR*, abs/1804.10173, 2018. arXiv:1804.10173.

27 Ross M McConnell and Jeremy P Spinrad. Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In *Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 536–545. Society for Industrial and Applied Mathematics, 1994.

28 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. Fine-grained algorithm design for matching. *CoRR*, abs/1609.08879, 2016. arXiv:1609.08879.

29 Silvio Micali and Vijay V. Vazirani. An o(sqrt(|v|) |e|) algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.12.

30 James B. Orlin. Max flows in o(nm) time, or better. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 765–774, 2013. doi:10.1145/2488608.2488705.

31 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1065–1075. SIAM, 2010. doi:10.1137/1.9781611973075.86.

32 Thomas Schank and Dorothea Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *Experimental and Efficient Algorithms, 4th InternationalWorkshop, WEA 2005, Santorini Island, Greece, May 10-13, 2005, Proceedings*, pages 606–609, 2005. doi:10.1007/11427186_54.

33 Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *J. ACM*, 44(4):585–591, 1997. doi:10.1145/263867.263872.

34 Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645. Springer, 2008. doi:10.1007/978-3-540-70575-8_52.

On Nondeterministic Derandomization of Freivalds' Algorithm: Consequences, Avenues and Algorithmic Progress

Marvin Künnemann
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
marvin@mpi-inf.mpg.de

Abstract

Motivated by studying the power of randomness, certifying algorithms and barriers for fine-grained reductions, we investigate the question whether the multiplication of two $n \times n$ matrices can be performed in near-optimal *nondeterministic* time $\tilde{\mathcal{O}}(n^2)$. Since a classic algorithm due to Freivalds verifies correctness of matrix products probabilistically in time $\mathcal{O}(n^2)$, our question is a relaxation of the open problem of derandomizing Freivalds' algorithm.

We discuss consequences of a positive or negative resolution of this problem and provide potential avenues towards resolving it. Particularly, we show that sufficiently fast deterministic verifiers for 3SUM or univariate polynomial identity testing yield faster deterministic verifiers for matrix multiplication. Furthermore, we present the partial algorithmic progress that distinguishing whether an integer matrix product is correct or contains between 1 and n erroneous entries can be performed in time $\tilde{\mathcal{O}}(n^2)$ – interestingly, the difficult case of deterministic matrix product verification is not a problem of "finding a needle in the haystack", but rather cancellation effects in the presence of many errors.

Our main technical contribution is a deterministic algorithm that corrects an integer matrix product containing at most t errors in time $\tilde{\mathcal{O}}(\sqrt{t}n^2 + t^2)$. To obtain this result, we show how to compute an integer matrix product with at most t nonzeroes in the same running time. This improves upon known deterministic output-sensitive integer matrix multiplication algorithms for $t = \Omega(n^{2/3})$ nonzeroes, which is of independent interest.

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

Keywords and phrases matrix product verification, certifying computation, fine-grained complexity and algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.56

Related Version A full version of the paper is available at [27], `https://arxiv.org/abs/1806.09189`.

Acknowledgements The author wishes to thank Markus Bläser, Russell Impagliazzo, Kurt Mehlhorn, Ramamohan Paturi, and Michael Sagraloff for early discussions on this work and Karl Bringmann for comments on a draft of this paper.

1 Introduction

Fast matrix multiplication algorithms belong to the most exciting algorithmic developments in the realm of low-degree polynomial-time problems. Starting with Strassen's polynomial speedup [38] over the naive $\mathcal{O}(n^3)$-time algorithm, extensive work (see, e.g., [13, 41, 29]) has brought down the running time to $\mathcal{O}(n^{2.373})$ (we refer to [8] for a survey). This leads to substantial improvements over naive solutions for a wide range of applications; for many

© Marvin Künnemann;
licensed under Creative Commons License CC-BY

problems, the best known algorithms make crucial use of fast multiplication of square or rectangular matrices. To name just a few examples, we do not only obtain polynomial improvements for numerous tasks in linear algebra (computing matrix inverses, determinants, etc.), graph theory (finding large cliques in graphs [33], All-Pairs Shortest Path for bounded edge-weights [4]), stringology (context free grammar parsing [40], RNA folding and language edit distance [9]) and many more, but also strong subpolynomial improvements such as a $2^{\Omega(\sqrt{\log n})}$-factor speed-up for the All-Pairs Shortest Path problem (APSP) [46] or similar improvements for the orthogonal vectors problem (OV) [3]. It is a famous open question whether the matrix multiplication exponent ω is equal to 2.

Matrix multiplication is the search version of the MM-Verification problem: given $n \times n$ matrices A, B and a candidate C for the product matrix, verify whether $AB = C$. There is a surprisingly simple randomized algorithm due to Freivalds [15] that is correct with probability at least 1/2: Pick a random vector $v \in \{0,1\}^n$, compute the matrix-vector products Cv and $A(Bv)$, and declare $AB = C$ if and only if $Cv = ABv$. Especially given the simplicity of this algorithm and the widely-shared hope that $\omega = 2$, one might conjecture that a deterministic version of Freivalds' algorithm exists. Alas, while refined ways to pick the random vector v reduce the required number of random bits to $\log n + \mathcal{O}(1)$ [32, 26], a $\tilde{\mathcal{O}}(n^2)$-time deterministic algorithms for matrix product verification remains elusive.

The motivation of this paper is the following question:

Can we solve Boolean, integer or real matrix multiplication in nondeterministic $\tilde{\mathcal{O}}(n^2)$ *time?*

Here we say that a functional problem f is in nondeterministic time $t(n)$ if f admits a $t(n)$*-time verifier*: there is a function v, computable in deterministic time $t(n)$, where n denotes the problem size of x, such that for all x, y there exists a certificate c with $v(x, y, c) = 1$ if and only $y = f(x)$.[1]

Note that a $\tilde{\mathcal{O}}(n^2)$-time derandomization of Freivalds' algorithm would yield an affirmative answer: guess C, and verify $AB = C$ using the deterministic verification algorithm. In contrast, a nondeterministic algorithm may guess additional information, a *certificate* beyond a guess C on the matrix product, and use it to verify that $C = AB$. Surprising faster algorithms in such settings have recently been found for 3SUM and all problems subcubic equivalent to APSP under deterministic reductions [11]; see [43, 42] for an overview over subcubic equivalences to APSP.

In this paper, we discuss consequences of positive or negative resolutions of this question, propose potential avenues for an affirmative answer and present partial algorithmic progress. In particular, we show that (1) sufficiently fast verifiers for 3SUM or univariate polynomial identity testing yield faster nondeterministic matrix multiplication algorithms, (2) in the integer case we can detect existence of between 1 and n erroneous entries in C in deterministic time $\tilde{\mathcal{O}}(n^2)$ and (3) we provide a novel deterministic output-sensitive integer matrix multiplication algorithm that improves upon previous deterministic algorithms if AB has at least $n^{2/3}$ nonzeroes.

1.1 Further Motivation and Consequences

Our motivation stems from studying the power of randomness, as well as algorithmic applications in certifiable computation, and consequences for the fine-grained complexity of polynomial-time problems.

[1] Throughout the paper, we view any decision problem P as a binary-valued functional problem. Thus a $t(n)$-time verifier for P shows that P is in nondeterministic *and* co-nondeterministic time $t(n)$.

Power of Randomness: Matrix-product verification has one of the simplest randomized solution for which no efficient derandomization is known – the currently best known deterministic algorithm simply computes the matrix product AB in deterministic time $\mathcal{O}(n^\omega)$ and checks whether $C = AB$. Exploiting nondeterminism instead of randomization may yield insights into when and under which conditions we can derandomize algorithms without polynomial increases in the running time.

A very related case is that of univariate polynomial identity testing (UPIT): it has a similar status with regards to randomized and deterministic algorithms. As we will see, finding $\tilde{\mathcal{O}}(n^2)$-time nondeterministic derandomizations for UPIT is a more difficult problem, so that resolving our main question appears to be a natural intermediate step towards nondeterministic derandomizations of UPIT, see Section 1.2.

Practical Applications – Deterministic Certifying Algorithms: Informally, a *certifying algorithm* for a functional problem f is an algorithm that computes, for each input x, besides the desired output $y = f(x)$ also a certificate c such that there is a *simple* verifier that checks whether c proves that $y = f(x)$ indeed holds [31]. If we fix our notion of *simplicity* to be that of being computable by a fast deterministic algorithm, then our notion of verifiers turns out to be a suitable notion to study existence of certifying algorithms – it only disregards the running time needed to compute the certificate c.

Having a fast verifier for matrix multiplication would certainly be desirable – while Freivalds' algorithm yields a solution that is sufficient for many practical applications, it can never *completely* remove doubts on the correctness. Since matrix multiplication is a central ingredient for many problems, fast verifiers for matrix multiplication imply fast verifiers for many more problems.

In fact, even if $\omega = 2$, finding *combinatorial*[2] strongly subcubic verifiers is of interest, as these are more likely to yield practical advantages over more naive solutions. In particular, the known subcubic verifiers for all problems subcubic equivalent to APSP (under deterministic reductions) [11] all rely on fast matrix multiplication, and might not yet be relevant for practical applications.

Barriers for SETH-based Lower Bounds: Given the widely-shared hope that $\omega = 2$, can we rule out conditional lower bounds of the form $n^{c-o(1)}$ with $c > 2$ for matrix multiplication, e.g., based on the Strong Exponential Time Hypothesis (SETH) [19]? Carmosino et al. [11] proposed the Nondeterministic Strong Exponential Time Hypothesis (NSETH) that effectively postulates that there is no $\mathcal{O}(2^{(1-\varepsilon)n})$-time co-nondeterministic algorithm for k-SAT for all constant k. Under this assumption, we can rule out fast nondeterministic or co-nondeterministic algorithms for all problems that have *deterministic* fine-grained reductions from k-SAT. Conversely, if we find a nondeterministic matrix multiplication algorithm running in time $n^{c+o(1)}$, then NSETH implies that there is no SETH-based lower bound of $n^{c'-o(1)}$, with $c' > c$, for matrix multiplication using deterministic reductions.

Barriers for Reductions in Case of a Negative *Resolution*: Suppose that there is a negative resolution of our main question, specifically that Boolean matrix multiplication has no $n^{c-o(1)}$-time verifier for some $c > 2$ (observe that this would imply $\omega > 2$). Then by a simple $\mathcal{O}(n^2)$-time nondeterministic reduction from Boolean matrix multiplication to triangle finding (implicit in the proof of Theorem 1.1 below) and a known $\mathcal{O}(n^2)$-time reduction from triangle finding to Radius [1], Radius has no $n^{c-o(1)}$-time verifier. This state of affairs would rule out certain kinds of subcubic reductions from Radius to Diameter, e.g., deterministic

[2] Throughout this paper, we call an algorithm *combinatorial*, if it does not use sophisticated algebraic techniques underlying the fastest known matrix multiplication algorithms.

many-one-reductions, since these would transfer a simple $\mathcal{O}(n^2)$-time verifier for Diameter[3] to Radius. Note that finding a subcubic reduction from Radius to Diameter is an open problem in the fine-grained complexity community [1].

1.2 Structural Results: Avenues Via Other Problems

We present two particular avenues for potential subcubic or even near-quadratic matrix multiplication verifiers: finding fast verifiers for either 3SUM or univariate polynomial identity testing.

3SUM

One of the core hypotheses in the field of hardness in P is the 3SUM problem [16]. Despite the current best time bound of $\mathcal{O}(n^2 \cdot \frac{\text{poly} \log \log n}{\log^2 n})$ [6, 12] being only slightly subquadratic, recently a strongly subquadratic verifier running in time $\tilde{\mathcal{O}}(n^{3/2})$ was found [11]. We have little indication to believe that this verification time is optimal; for the loosely related computational model of decision trees, a remarkable near-linear time bound has been obtained just this year [25].

By a simple reduction, we obtain that any polynomial speedup over the known 3SUM verifier yields a subcubic Boolean matrix multiplication verifier. In particular, establishing a near-linear 3SUM verifier would yield a positive answer to our main question in the Boolean setting.

▶ **Theorem 1.1.** *Any $\mathcal{O}(n^{3/2-\varepsilon})$-time verifier for* 3SUM *yields a $\mathcal{O}(n^{3-2\varepsilon})$-time verifier for Boolean matrix multiplication.*

Under the BMM hypothesis, which asserts that there is no combinatorial $\mathcal{O}(n^{3-\varepsilon})$-time algorithm for Boolean matrix multiplication (see, e.g., [2]), a $n^{3/2-o(1)}$-time lower bound (under randomized reductions) for combinatorial 3SUM algorithms is already known [22, 43]. The above result, however, establishes a stronger, non-randomized relationship between the verifiers' running times by a simple proof exploiting nondeterminism.

UPIT

Univariate polynomial identity testing (UPIT) asks to determine, given two degree-n polynomials p, q over a finite field of polynomial order, represented as arithmetic circuits with $\mathcal{O}(n)$ wires, whether p is identical to q. By evaluating and comparing p and q at $n+1$ distinct points or $\tilde{\mathcal{O}}(1)$ random points, we can solve UPIT deterministically in time $\tilde{\mathcal{O}}(n^2)$ or with high probability in time $\tilde{\mathcal{O}}(n)$, respectively. A nondeterministic derandomization, more precisely, a $\mathcal{O}(n^{2-\varepsilon})$-time verifier, would have interesting consequences [47]: it would refute the Nondeterministic Strong Exponential Time Hypothesis posed by Carmosino et al. [11], which in turn would prove novel circuit lower bounds, deemed difficult to prove. We observe that a sufficiently strong nondeterministic derandomization of UPIT would also give a faster matrix multiplication verifier.

[3] We verify that a graph G has diameter d as follows: For every vertex v, we guess the shortest path tree originating in v. It is straightforward to use this tree to verify that all vertices v' have distance at most d from v in time $\mathcal{O}(n)$. Thus, we can prove that the diameter is at most d in time $\mathcal{O}(n^2)$. For the lower bound, guess some vertex pair u, v and verify that their distance is indeed d using a single-source shortest path computation in time $\mathcal{O}(m + n \log n) = \mathcal{O}(n^2)$.

▶ **Theorem 1.2.** *Any* $\mathcal{O}(n^{3/2-\varepsilon})$*-time verifier for* UPIT *yields a* $\mathcal{O}(n^{3-2\varepsilon})$*-time verifier for integer matrix multiplication.*

Note that this avenue might seem more difficult to pursue than a direct attempt at resolving our main question, due to its connection to NSETH and circuit lower bounds. Alternatively, however, we can view the specific arithmetic circuit obtained in our reductions as an interesting intermediate testbed for ideas towards derandomizing UPIT. In fact, our algorithmic results were obtained by exploiting the connection to UPIT, and exploiting the structure of the resulting specialized circuits/polynomials.

1.3 Algorithmic Results: Progress on Integer Matrix Product Verification

Our main result is partial algorithmic progress towards the conjecture in the integer setting. Specifically, we consider a restriction of MM-Verification to the case of detecting a bounded number t of errors. Formally, let MM-Verification$_t$ denote the following problem: given $n \times n$ integer matrices A, B, C with polynomially bounded entries, produce an output "$C = AB$" or "$C \neq AB$", where the output must always be correct if C and AB differ in at most t entries.

Our main result is an algorithm that solves MM-Verification$_t$ in near-quadratic time for $t = \mathcal{O}(n)$ and in strongly subcubic time for $t = \mathcal{O}(n^c)$ with $c < 2$.

▶ **Theorem 1.3.** *For any* $1 \leq t \leq n^2$*,* MM-Verification$_t$ *can be solved deterministically in time* $O((n^2 + tn)\log^{2+o(1)} n)$*.*

Interestingly, this shows that detecting the presence of very few errors is not a difficult case. Instead of a needle-in-the-haystack problem, we rather need to find a way to deal with cancellation effects in the presence of at least $\Omega(n)$ errors.

As a corollary, we obtain a different near-quadratic-time randomized algorithm for MM-Verification than Freivalds' algorithm: Run the algorithm of Theorem 1.3 for $t = n$ in time $\tilde{\mathcal{O}}(n^2)$. Afterwards, either $C = AB$ holds or C has at least $\Omega(n)$ erroneous entries. Thus it suffices to sample $\Theta(n)$ random entries i, j and to check whether $C_{i,j} = (AB)_{i,j}$ for all sampled entries (by naive computation of $(AB)_{i,j}$ in time $\mathcal{O}(n)$ each) to obtain an $\tilde{\mathcal{O}}(n^2)$-time algorithm that correctly determines $C = AB$ or $C \neq AB$ with constant probability. Potentially, this alternative to Freivalds' algorithm might be simpler to derandomize.

Finally, our algorithm for *detecting* up to t errors can be extended to a more involved algorithm that also *finds* all erroneous entries (if no more than t errors are present) and *correct* them in time $\tilde{\mathcal{O}}(\sqrt{t}n^2 + t^2)$. In fact, this problem turns out to be equivalent to the notion of output-sensitive matrix multiplication os-MM$_t$: Given $n \times n$ matrices A, B of polynomially bounded integer entries with the promise that AB contains at most t nonzeroes, compute AB.

▶ **Theorem 1.4.** *Let* $1 \leq t \leq n^2$*. Given* $n \times n$ *matrices* A, B, C *of polynomially bounded integers, with the property that* C *differs from* AB *in at most* t *entries, we can compute* AB *in time* $\mathcal{O}(\sqrt{t}n^2 \log^{2+o(1)} n + t^2 \log^{3+o(1)} n)$*. Equivalently, we can solve* os-MM$_t$ *in time* $\mathcal{O}(\sqrt{t}n^2 \log^{2+o(1)} n + t^2 \log^{3+o(1)} n)$*.*

Previous work by Gasieniec et al. [17] gives a $\tilde{\mathcal{O}}(n^2 + tn)$ *randomized* solution, as well as a $\tilde{\mathcal{O}}(tn^2)$ deterministic solution. Because of the parameter-preserving equivalence between t error correction and os-MM$_t$, this task is also solved by the randomized $\tilde{\mathcal{O}}(n^2 + tn)$-time algorithm due to Pagh [34][4] and the deterministic $\mathcal{O}(n^2 + t^2 n \log^5 n)$-time algorithm due to Kutzkov [28]. Note that our algorithm improves upon Kutzkov's algorithm for $t = \Omega(n^{2/3})$, in particular, our algorithm is strongly subcubic for $t = \mathcal{O}(n^{3/2-\varepsilon})$ and even improves upon the best known fast matrix multiplication algorithm for $t = \mathcal{O}(n^{0.745})$.

1.4 Further Related Work

There is previous work that claims to have resolved our main question in the affirmative. Unfortunately, the approach is flawed; we detail the issue in the full version of this article [27].

Other work considers MM-Verification and os-MM in quantum settings, e.g., [10, 23]. Furthermore, better running times can be obtained if we restrict the distribution of the errors over the guessed matrix/nonzeroes over the matrix product: Using rectangular matrix multiplication, Iwen and Spencer [20] show how to compute AB in time $\mathcal{O}(n^{2+\varepsilon})$ for any $\varepsilon > 0$, if no column (or no row) of AB contains more than $n^{0.29462}$ nonzeroes. Furthermore, Roche [35] gives a randomized algorithm refining the bound of Gasieniec et al. [17] using, as additional parameters, the total number of nonzeroes in A, B, C and the number of distinct columns/rows containing an error.

For the case of Boolean matrix multiplication, several output-sensitive algorithms are known [36, 48, 5, 30], including a simple deterministic $\mathcal{O}(n^2 + tn)$-time algorithm [36] and, exploiting fast matrix multiplication, a randomized $\tilde{\mathcal{O}}(n^2 t^{\omega/2-1})$-time solution [30]. Note that in the Boolean setting, our parameter-preserving reduction from error correction to output-sensitive multiplication (Proposition 3.1) no longer applies, so that these algorithms unfortunately do not immediately yield error correction algorithms.

1.5 Paper Organization

After collecting notational conventions and introducing polynomial multipoint evaluation as our main algorithmic tool in Section 2, we give a high-level description over the main ideas behind our results in Section 3. We prove our structural results in Section 4. Our first algorithmic result on error detection is proven in Section 5. Unfortunately, the details for our technically most demanding result, i.e., Theorem 1.4, had to be omitted due to space constraints – they are available in the full version of this article [27]. We conclude with open questions in Section 6.

2 Preliminaries

Recall the definition of a $t(n)$-time verifier for a functional problem f: there is a function v, computable in deterministic time $t(n)$ with n being the problem size of x, such that for all x, y there exists a certificate c with $v(x, y, c) = 1$ if and only $y = f(x)$. Here, we assume the word RAM model of computation with a word size $w = \Theta(\log n)$.

For n-dimensional vectors a, b over the integers, we write their inner product as $\langle a, b\rangle = \sum_{k=1}^{n} a[k] \cdot b[k]$, where $a[k]$ denotes the k-th coordinate of a. For any matrix X, we write $X_{i,j}$ for its value at row i, column j. We typically represent the $n \times n$ matrix A by its n-dimensional row vectors $a_1, \dots, a_n$, and the $n \times n$ matrix B by its n-dimensional column vectors $b_1, \dots, b_n$ such that $(AB)_{i,j} = \langle a_i, b_j\rangle$. For any $I \subseteq [n], J \subseteq [n]$, we obtain a submatrix $(AB)_{I,J}$ of AB by deleting from AB all rows not in I and all columns not in J.

[4] For $t = \omega(n)$, Jacob and Stöckel [21] give an improved randomized $\tilde{\mathcal{O}}(n^2 (t/n)^{\omega-2})$-time algorithm.

Fast Polynomial Multipoint Evaluation

Consider any finite field $\mathbb{F}$ and let $M(d)$ be the number of additions and multiplications in $\mathbb{F}$ needed to multiply two degree-d univariate polynomials. Note that $M(d) = \mathcal{O}(d \log d \log\log d) = \mathcal{O}(d \log^{1+o(1)} n)$, see, e.g. [44].

▶ **Lemma 2.1** (Multipoint Polynomial Evaluation [14])**.** *Let $\mathbb{F}$ be an arbitrary field. Given a degree-d polynomial $p \in \mathbb{F}[X]$ given by a list of its coefficients $(a_0, \dots, a_d) \in \mathbb{F}^{d+1}$, as well as input points $x_1, \dots, x_d \in \mathbb{F}$, we can determine the list of evaluations $(p(x_1), \dots, p(x_d)) \in \mathbb{F}^n$ using $\mathcal{O}(M(d) \log d)$ additions and multiplications in $\mathbb{F}$.*

Thus, we can evaluate p on any list of inputs $x_1, \dots, x_n$ in time $\mathcal{O}((n + d) \log^{2+o(1)} d)$.

3 Technical Overview

We first observe a simple parameter-preserving equivalence of the following problems,

MM-Verification$_t$ Given $\ell \times n, n \times \ell, \ell \times \ell$ matrices A, B, C such that AB and C differ in $0 \le z \le t$ entries, determine whether $AB = C$, i.e., $z = 0$,

AllZeroes$_t$ Given $\ell \times n, n \times \ell$ matrices A, B such that AB has $0 \le z \le t$ nonzeroes, determine whether $AB = 0$, i.e., $z = 0$.

We also obtain a parameter-preserving equivalence of their "constructive" versions,

MM-Correction$_t$ Given $\ell \times n, n \times \ell, \ell \times \ell$ matrices A, B, C such that AB and C differ in $0 \le z \le t$ entries, determine AB,

os-MM$_t$ Given $\ell \times n, n \times \ell$ matrices A, B such that AB has $0 \le z \le t$ nonzeroes, determine AB.

For any problem P_t among the above, let $T_P(n, \ell, t)$ denote the optimal running time to solve P_t with parameters n, ℓ and t.

▶ **Proposition 3.1.** *Let $\ell \le n$ and $1 \le t \le n^2$. We have*

$$T_{\text{MM-Verification}}(n, \ell, t) = \Theta(T_{\text{AllZeroes}}(n, \ell, t))$$
$$T_{\text{MM-Correction}}(n, \ell, t) = \Theta(T_{\text{os-MM}}(n, \ell, t)).$$

Proof. By setting $C = 0$, we can reduce AllZeroes$_t$ and os-MM$_t$ to MM-Verification$_t$ and MM-Correction$_t$, respectively, achieving the lower bounds of the claim.

For the other direction, let $a_1, \dots, a_\ell \in \mathbb{Z}^n$ be the row vectors of A, $b_1, \dots, b_\ell \in \mathbb{Z}^n$ be the column vectors of B and $c_1, \dots, c_\ell \in \mathbb{Z}^\ell$ be the column vectors of C. Let e_i denote the vector whose i-th coordinate is 1 and whose other coordinates are 0. We define $\ell \times (n+\ell), (n+\ell) \times \ell$ matrices A', B' by specifying the row vectors of A' as

$$a'_i = (a_i, -e_i),$$

and the column vectors of B' as

$$b'_j = (b_j, c_j).$$

Note that $(A'B')_{i,j} = \langle a'_i, b'_j \rangle = \langle a_i, b_j \rangle - c_j[i]$, thus $(A'B')_{i,j} = 0$ if and only if $(AB)_{i,j} = C_{i,j}$. Consequently, $A'B'$ has at most t nonzeroes, and checking equality of $A'B'$ to the all-zero matrix is equivalent to checking $AB = C$. The total time to solve MM-Verification$_t$ is thus bounded by $\mathcal{O}((n+\ell)\ell) + T_{\text{AllZeroes}}(n+\ell, \ell, t) = \mathcal{O}(T_{\text{AllZeroes}}(n, \ell, t))$, as desired.

Furthermore, by computing $C' = A'B'$ (which contains at most t nonzero entries), we can also correct the matrix product C by updating $C_{i,j}$ to $C_{i,j} + C'_{i,j}$. This takes time $\mathcal{O}((n+\ell)\ell) + T_{\text{os-MM}}(n+\ell, \ell, t) = \mathcal{O}(T_{\text{os-MM}}(n, \ell, t))$, as desired. ◀

Because of the above equivalence, we can focus on solving AllZeroes$_t$ and os-MM$_t$ in the remainder of the paper. The key for our approach is the following multilinear polynomial

$$f_{\mathrm{MM}}^{A,B}(x_1,\dots,x_\ell;y_1,\dots,y_\ell) := \sum_{i,j\in[\ell]} x_i \cdot y_j \cdot \langle a_i, b_j\rangle,$$

where again the $a_1,\dots,a_\ell$ denote the row vectors of A and the $b_1,\dots,b_\ell$ denote the column vectors of B. Note that the nonzero monomials of $f_{\mathrm{MM}}^{A,B}$ correspond directly to the nonzero entries of AB. We introduce a univariate variant

$$g(X) = g^{A,B}(X) := f_{\mathrm{MM}}^{A,B}(1, X, \dots, X^{\ell-1}; 1, X^\ell, \dots, X^{\ell(\ell-1)}),$$

which has the helpful property that monomials $x_i y_j$ of f_{MM} are mapped to the monomial $X^{(i-1)+\ell(j-1)}$ in a one-to-one manner, preserving coefficients. To obtain a more efficient representation of g than to explicitly compute all coefficients $\langle a_i, b_j\rangle$, we can exploit linearity of the inner product: we have $g(X) = \sum_{k=1}^n q_k(X) r_k(X^\ell)$, where $q_k(Z) = \sum_{i=1}^\ell a_i[k] Z^{i-1}$ and $r_k(Z) = \sum_{j=1}^\ell b_j[k] Z^{j-1}$. This representation is more amenable for efficient evaluation, and immediately yields a reduction to univariate polynomial identity testing (UPIT) (see Theorem 4.2 in Section 4).

To solve the detection problem, we use an idea from sparse polynomial interpolation [7, 49]: If AB has at most t nonzeroes, then for any root of unity ω of sufficiently high order, $g(\omega^0) = g(\omega^1) = g(\omega^2) = \cdots = g(\omega^{t-1}) = 0$ is equivalent to $AB = 0$. By showing how to do fast batch evaluation of g using the above representation, we obtain an $\tilde{\mathcal{O}}((\ell+t)n)$-time algorithm for AllZeroes$_t$ in Section 5, proving Theorem 1.3.

Towards solving the correction problem, the naive approach is to use the $\tilde{\mathcal{O}}((\ell+t)n)$-time AllZeroes$_t$ algorithm in combination with a self-reduction to obtain a fast algorithm for finding a nonzero position (i,j) of AB: If the AllZeroes algorithm determines that AB contains at least one nonzero entry, we split the product matrix AB into four submatrices, detect any one of them containing a nonzero entry, and recurse on it. After finding such an entry, one can compute the correct nonzero value $(AB)_{i,j} = \langle a_i, b_j\rangle$ in time $\mathcal{O}(n)$. One can then "remove" this nonzero from further search (analogously to Proposition 3.1) and iterate this process. Unfortunately, this only yields an algorithm of running time $\tilde{\mathcal{O}}(tn^2)$, even if AllZeroes would take near-optimal time $\tilde{\mathcal{O}}(n^2)$. A faster alternative is to use the self-reduction such that we find *all* nonzero entries whenever we recurse on a submatrix containing at least one nonzero value. However, this process only leads to a running time of $\tilde{\mathcal{O}}(\sqrt{t}n^2 + nt^2)$. Here, the bottleneck $\tilde{\mathcal{O}}(nt^2)$ term stems from the fact that performing an AllZeroes test for t submatrices (e.g., when t nonzeroes are spread evenly in the matrix) takes time $t \cdot \tilde{\mathcal{O}}(nt)$.

We still obtain a faster algorithm by a rather involved approach: The intuitive idea is to test submatrices for appropriately smaller number of nonzeroes $z \ll t$. At first sight, such an approach might seem impossible, since we can only be certain that a submatrix contains no nonzeroes if we test it for the full number t of potential nonzeroes. However, by showing how to reuse and quickly update previously computed information after finding a nonzero, we make this approach work by obtaining "global" information at a small additional cost of $\tilde{\mathcal{O}}(t^2)$. Doing these dynamic updates quickly crucially relies on the efficient representation of the polynomial g. The details are given in the full version of this article [27].

4 Structural Results: Avenues Via Other Problems

In this section, we show the simple reductions translating verifiers for 3SUM or UPIT to matrix multiplication.

4.1 3SUM

We consider the following formulation of the 3SUM problem: given sets S_1, S_2, S_3 of polynomially bounded integers, determine whether there exists a triplet $s_1 \in S_1, s_2 \in S_2, s_3 \in S_3$ with $s_1 + s_2 = s_3$. It is known that a combinatorial $\mathcal{O}(n^{3/2-\varepsilon})$-time algorithm for 3SUM (for any $\varepsilon > 0$) yields a combinatorial $\mathcal{O}(n^{3-\varepsilon'})$-time Boolean matrix multiplication (BMM) algorithm (for some $\varepsilon' > 0$). This follows by combining a reduction from Triangle Detection to 3SUM of [22] and using the combinatorial subcubic equivalence of Triangle Detection and BMM [43][5]. While this only yields a *nontight* BMM-based lower bound for 3SUM for *deterministic or randomized* combinatorial algorithms, we can establish a *tight* relationship for the current state of knowledge of combinatorial verifiers. In fact, allowing nondeterminism, we obtain a very simple direct proof of a stronger relationship of the running times than known for deterministic reductions.

▶ **Theorem 4.1.** *If 3SUM admits a ("combinatorial") $\mathcal{O}(n^{3/2-\varepsilon})$-time verifier, then* BMM *admits a ("combinatorial") $\mathcal{O}(n^{3-2\varepsilon})$-time verifier.*[6]

Thus, significant combinatorial improvements over Carmosino et al.'s 3SUM verifier yield strongly subcubic combinatorial BMM verifiers. In particular, a $\tilde{\mathcal{O}}(n)$-time verifier for 3SUM would yield an affirmative answer to our main question in the Boolean setting. Note that an analogous improvement of the $\mathcal{O}(n^{3/2}\sqrt{\log n})$ [18] size bound in the decision tree model to a size of $\mathcal{O}(n \log^2 n)$ has recently been obtained [25].

To establish this strong relationship, our reduction exploits the nondeterministic setting – without nondeterminism, no reduction is known that would give a $\mathcal{O}(n^{\frac{8}{3}-\varepsilon})$-time BMM algorithm even if 3SUM could be solved in an optimal $\mathcal{O}(n)$ time bound.

Proof of Theorem 4.1. Given the $n \times n$ Boolean matrices A, B, C, we first check whether all entries (i, j) with $C_{i,j} = 1$ are correct. For this, for each such i, j, we guess a witness k and check that $A_{i,k} = B_{k,j} = 1$, which verifies that $C_{i,j} = (AB)_{i,j} = 1$.

To check the remaining zero entries $Z = \{(i, j) \in [n]^2 \mid C_{i,j} = 0\}$, we construct a 3SUM instance S_1, S_2, S_3 as follows. Let $W = 2(n + 1)$. For each $(i, j) \in Z$, we include $iW^2 + jW$ in our set S_3. For every (i, k) with $A_{i,k} = 1$, we include $iW^2 + k$ in our set S_1, and, for every (k, j) with $B_{k,j} = 1$, we include $jW - k$ in our set S_2. Clearly, any witness $A_{i,k} = B_{k,j} = 1$ for $(AB)_{i,j} = 1$, $(i, j) \in Z$ yields a triplet $a = iW^2 + k \in S_1, b = jW - k \in S_2, c = iW^2 + jW \in S_3$ with $a + b = c$. Conversely, any 3SUM triplet $a \in S_1, b \in S_2, c \in S_3$ yields a witness for $(AB)_{i,j} = 1$, where $(i, j) \in Z$ is the zero entry represented by c, since $(iW^2 + k) + (jW - k') = i'W^2 + j'W$ for $i, i', j, j', k, k' \in [n]$ if only if $i = i', j = j'$ and $k = k'$ by choice of W. Thus, the 3SUM instance is a NO instance if and only if no $(i, j) \in Z$ has a witness for $(AB)_{i,j} = 1$, i.e., all $(i, j) \in Z$ satisfy $C_{i,j} = (AB)_{i,j} = 0$.

Note that reduction runs in nondeterministic time $\mathcal{O}(n^2)$, using an oracle call of a 3SUM instance of size $\mathcal{O}(n^2)$, which yields the claim. ◀

[5] K. G. Larsen obtained an independent proof of this fact, see `https://simons.berkeley.edu/talks/kasper-larsen-2015-12-01`.

[6] Strictly speaking, the notion of a "combinatorial" algorithm is not well-defined, hence we use quotes here. However, our reductions are so simple that they should qualify under any reasonable exact definition.

4.2 UPIT

Univariate Polynomial Identity Testing (UPIT) is the following problem: Given arithmetic circuits Q, Q' on a single variable, with degree n and $O(n)$ wires, over a field of order $\mathrm{poly}(n)$, determine whether $Q \equiv Q'$, i.e., the outputs of Q and Q' agree on all inputs. Using evaluation on $n+1$ distinct points, we can deterministically solve UPIT in time $\tilde{\mathcal{O}}(n^2)$, while evaluating on $\tilde{\mathcal{O}}(1)$ random points yields a randomized solution in time $\tilde{\mathcal{O}}(n)$. Williams [47] proved that a $\mathcal{O}(n^{2-\varepsilon})$-time deterministic UPIT algorithm refutes the Nondeterministic Strong Exponential Time Hypothesis posed by Carmosino et al. [11]. We establish that a sufficiently strong (nondeterministic) derandomization of UPIT also yields progress on MM-Verification.

▶ **Theorem 4.2.** *If* UPIT *admits a ("combinatorial") $\mathcal{O}(n^{3/2-\varepsilon})$-time verifier for some $\varepsilon > 0$, then there is a ("combinatorial") $\mathcal{O}(n^{3-2\varepsilon})$-time verifier for matrix multiplication over polynomially bounded integers and over finite fields of polynomial order.*

Proof. We only give the proof for matrix multiplication over a finite field $\mathbb{F}$ of polynomial order. Using Chinese Remaindering, we can easily extend the reduction to the integer case (see Proposition 5.3 below).

Consider $g(X) = \sum_{i,j\in[n]} \langle a_i, b_j\rangle X^{(i-1)+n(j-1)}$ over $\mathbb{F}$ as defined in Section 3 (with $\ell = n$). As described there, we can write $g(X) = \sum_{k=1}^{n} q_k(X) r_k(X^n)$ with $q_k(Z) = \sum_{i=1}^{n} a_i[k] Z^{i-1}$ and $r_k(Z) = \sum_{j=1}^{n} b_j[k] Z^{j-1}$. Let $k \in [n]$ and note that q_k, r_k and X^n have arithmetic circuits with $\mathcal{O}(n)$ wires using Horner's scheme. Chaining the circuits of X^n and r_k, and multiplying with the output of the circuit for q_k, we obtain a degree-$\mathcal{O}(n^2)$ circuit Q_k with $\mathcal{O}(n)$ wires. It remains to sum up the outputs of the circuits $Q_1, \dots, Q_n$. We thus obtain a circuit Q with $\mathcal{O}(n^2)$ wires and degree $\mathcal{O}(n^2)$. Since by construction $AB = 0$ if and only $Q \equiv 0$, we obtain an UPIT instance Q, Q', with Q' being a constant-sized circuit with output 0, that is equivalent to our MM-Verification instance. Thus, any $\mathcal{O}(n^{3/2-\varepsilon})$-time algorithm for UPIT would yield a $\mathcal{O}(n^{2(3/2-\varepsilon)})$-time MM-Verification algorithm, as desired. ◀

It is known that refuting NSETH implies strong circuit lower bounds [11], so pursuing this route might seem much more difficult than attacking MM-Verification directly. However, to make progress on MM-Verification, we only need to nondeterministically derandomize UPIT for very specialized circuits. In this direction, our algorithmic results exploit that we can derandomize UPIT for these specialized circuits, as long as they represent *sparse* polynomials.

5 Deterministically Detecting Presence of $0 < z \le t$ Errors

In this section we prove the first of our main algorithmic results, i.e., Theorem 1.3.

▶ **Theorem 5.1.** *For any $1 \le t \le n^2$,* MM-Verification$_t$ *can be solved deterministically in time $O((n^2 + tn)\log^{2+o(1)}(n))$.*

We prove the claim by showing how to solve the following problem in time $\tilde{\mathcal{O}}((\ell + t)n)$.

▶ **Lemma 5.2.** *Let $\mathbb{F}_p$ be a prime field with a given element $\omega \in \mathbb{F}_p$ of order at least ℓ^2. Let A, B be $\ell \times n, n \times \ell$-matrices over $\mathbb{F}_p$. There is an algorithm running in time $\mathcal{O}((\ell + t)n\log^{2+o(1)} n)$ with the following guarantees:*

1. *If $AB = 0$, the algorithm outputs "$AB = 0$".*
2. *If AB has $0 < z \le t$ nonzeroes, the algorithm outputs "$AB \neq 0$".*

Given such an algorithm working over finite fields, we can check matrix products of integer matrices using the following proposition.

▶ **Proposition 5.3.** *Let A, B be $n \times n$ matrices over the integers of absolute values bounded by n^c for some $c \in \mathbb{N}$. Then we can find, in time $\mathcal{O}(n^2 \log n)$, distinct primes $p_1, p_2, \dots, p_d$ and corresponding elements $\omega_1 \in \mathbb{F}_{p_1}, \omega_2 \in \mathbb{F}_{p_2}, \dots, \omega_d \in \mathbb{F}_{p_d}$, such that*

i) *$AB = 0$ if and only if $AB = 0$ over $\mathbb{F}_{p_i}$ for all $1 \le i \le d$,*
ii) *$d = \mathcal{O}(1)$, and*
iii) *for each $1 \le i \le d$, we have $p_i = \mathcal{O}(n^2)$ and ω_i has order at least n^2 in $\mathbb{F}_{p_i}$.*

Note that the obvious approach of choosing a single prime field $\mathbb{F}_p$ with $p \ge n^{2c+1}$ is not feasible for our purposes: the best known deterministic algorithm to find such a prime takes time $n^{c/2+o(1)}$ (see [39] for a discussion), quickly exceeding our desired time bound of $\mathcal{O}(n^2)$.

Proof of Proposition 5.3. Let $d = c + 1$ and note that any entry $(AB)_{i,j} = \sum_{k=1}^n A_{i,k}B_{k,j}$ is in $[-n^{2c+1}, n^{2c+1}]$. Thus for any number $m > n^{2c+1}$, we have $(AB)_{ij} \equiv 0 \pmod m$ if and only if $(AB)_{i,j} = 0$. By Chinese Remaindering, we obtain that any distinct primes $p_1, \dots, p_d$ with $p_i \ge n^2$ satisfy i) and ii), as $AB = 0$ if and only if $AB = 0$ over $\mathbb{F}_{p_i}$ for all $1 \le i \le d$, using the fact that $\prod_{i=1}^d p_i \ge n^{2d} > n^{2c+1}$.

By Bertrand's postulate, there are at least d primes in the range $\{n^2 + 1, \dots, 2^d(n^2 + 1)\}$, thus using the sieve of Eratosthenes, we can find $p_1, \dots, p_d$ with $p_i \ge n^2 + 1$ and $p_i \le 2^d(n^2 + 1)$ in time $\mathcal{O}(n^2 \log \log n)$ (see [44, Theorem 18.10]). It remains to find elements $\omega_1 \in \mathbb{F}_{p_1}, \dots, \omega_d \in \mathbb{F}_{p_d}$ of sufficiently high order. For each $1 \le j \le d$, this can be achieved in time $\mathcal{O}(n^2 \log n)$ by exhaustive testing: We keep a list $L \subseteq \mathbb{F}_{p_j}^\times = \mathbb{F}_{p_j} \setminus \{0\}$ of "unencountered" elements, which we initially set to $\mathbb{F}_{p_j}^\times$. Until there are no elements in L remaining, we pick any $\alpha \in L$ and delete all elements in the subgroup of $\mathbb{F}_{p_j}^\times$ generated by α from L. We set ω_j to the last α that we picked (which has to generate the complete multiplicative group $\mathbb{F}_{p_j}^\times$) and thus is a primitive $(p_j - 1)$-th root of unity. Since $p_j - 1 \ge n^2$, the order of ω_j is at least n^2, as desired. Observe that the number of iterations is bounded by the number of subgroups of $\mathbb{F}_{p_j}^\times$, i.e., the number of divisors of $p_j - 1$. Thus, we have at most $\mathcal{O}(\log p_j)$ iterations, each taking time at most $\mathcal{O}(p_j)$, yielding a running time of $\mathcal{O}(p_j \log p_j) = \mathcal{O}(n^2 \log n)$. ◀

Combining Proposition 3.1 with the algorithm of Lemma 5.2 and Proposition 5.3, we obtain the theorem.

Proof of Theorem 5.1. Given any instance A, B, C of MM-VERIFICATION$_t$, we convert it to an instance A', B' of ALLZEROES as in Proposition 3.1. We construct primes $p_1, \dots, p_d$ as in Proposition 5.3 in time $\mathcal{O}(n^2 \log n)$. For each $j \in [d]$, we convert A', B' to matrices over $\mathbb{F}_{p_j}$ in time $\mathcal{O}(n^2)$ and test whether $A'B' = 0$ over $\mathbb{F}_{p_j}$ for all $j \in [d]$ using Lemma 5.2 in time $\mathcal{O}((n^2 + tn) \log^{2+o(1)} n)$. We output "$AB = C$" if and only if all tests succeeded. Correctness follows from Proposition 5.3 and Lemma 5.2, and the total running time is $\mathcal{O}((n^2 + tn) \log^{2+o(1)} n)$, as desired. ◀

In the remainder, we prove Lemma 5.2. As outlined in Section 3, define the polynomial $g(X) = \sum_{i,j \in [\ell]} \langle a_i, b_j \rangle X^{(i-1)+\ell(j-1)}$ over $\mathbb{F}_p$. We aim to determine whether $g \equiv 0$. To do so, we use the following idea from Ben-Or and Tiwari's approach to black-box sparse polynomial interpolation (see [7, 49]). Suppose that $\omega \in \mathbb{F}_p$ has order at least ℓ^2. Then the following proposition holds.

▶ **Proposition 5.4.** *Assume AB has $0 \le z \le t$ nonzeroes. Then $g(\omega^0) = g(\omega) = g(\omega^2) = \dots = g(\omega^{t-1}) = 0$ if and only if $g \equiv 0$, i.e., $z = 0$.*

Proof. By assumption on A, B, we have $g(X) = \sum_{m \in M} c_m X^m$, where $M = \{(i-1)+\ell(j-1) \mid \langle a_i, b_j \rangle \neq 0\}$ with $|M| = z \leq t$ and $c_{(i-1)+\ell(j-1)} = \langle a_i, b_j \rangle$. Writing $M = \{m_1, \dots, m_z\}$ and defining $v_m = \omega^m$, we see that $g(\omega^0) = \cdots = g(\omega^{t-1}) = 0$ is equivalent to

$$\begin{aligned}
c_{m_1} &+ \cdots + c_{m_z} &&= 0,\\
c_{m_1} v_{m_1} &+ \cdots + c_{m_z} v_{m_z} &&= 0,\\
c_{m_1} v_{m_1}^2 &+ \cdots + c_{m_z} v_{m_z}^2 &&= 0,\\
& &&\dots\\
c_{m_1} v_{m_1}^{t-1} &+ \cdots + c_{m_z} v_{m_z}^{t-1} &&= 0.
\end{aligned}$$

Since ω has order at least ℓ^2, we have that $v_m = \omega^m \neq \omega^{m'} = v_{m'}$ for all $m, m' \in M$ with $m \neq m'$. Thus the above system is a Vandermonde system with unique solution $(c_{m_1}, \dots, c_{m_z}) = (0, \dots, 0)$, since $z \leq t$. This yields the claim. ◀

It remains to compute $g(\omega^0), \dots, g(\omega^{t-1})$ in time $\tilde{\mathcal{O}}((\ell + t)n)$.

▶ **Proposition 5.5.** *For any $\sigma_1, \dots, \sigma_t \in \mathbb{F}_p$, we can compute $g(\sigma_1), \dots, g(\sigma_t)$ in time $\mathcal{O}((\ell + t)n \log^{2+o(1)} \ell)$.*

Proof. Recall that $g(X) = \sum_{k=1}^n q_k(X) \cdot r_k(X^\ell)$, where $q_k(Z) = \sum_{i=1}^\ell a_i[k] Z^{i-1}$ and $r_k(Z) = \sum_{j=1}^\ell b_j[k] Z^{j-1}$. Let $1 \leq k \leq n$. Using fast multipoint evaluation (Lemma 2.1), we can compute $q_k(\sigma_1), \dots, q_k(\sigma_t)$ using $\mathcal{O}((\ell + t) \log^{2+o(1)} \ell)$ additions and multiplications in $\mathbb{F}_p$. Furthermore, since we can compute $\sigma_1^\ell, \dots, \sigma_t^\ell$ using $\mathcal{O}(t \log \ell)$ additions and multiplications in $\mathbb{F}_p$, we can analogously compute $r_k(\sigma_1^\ell), \dots, r_k(\sigma_t^\ell)$ in time $\mathcal{O}((\ell + t) \log^{2+o(1)} \ell)$. Doing this for all $1 \leq k \leq n$ yields all values $q_k(\sigma_u), r_k(\sigma_u^\ell)$ with $k \in [n], u \in [t]$ in time $\mathcal{O}((\ell + t)n \log^{2+o(1)} \ell)$. We finally aggregate these values to obtain the desired outputs $g(\sigma_u) = \sum_{k=1}^n q_k(\sigma_u) \cdot r_k(\sigma_u^\ell)$ with $u \in [t]$. The aggregation only uses $\mathcal{O}(tn)$ multiplications and additions in $\mathbb{F}_p$, thus the claim follows. ◀

Together with Proposition 5.4, this yields Lemma 5.2 and thus the remaining step of the proof of Theorem 5.1.

6 Open Questions

It remains to answer our main question. To this end, can we exploit any of the avenues presented in this work? In particular: Can we (1) find a faster 3SUM verifier, (2) find a faster UPIT algorithm for the circuits given in Theorem 4.2, or (3) instead of derandomizing Freivalds' algorithm, nondeterministically derandomize the sampling-based algorithm following from our main algorithmic result (which detects up to $\mathcal{O}(n)$ errors using Theorem 1.3, and then samples and checks $\Theta(n)$ random entries)?

A further natural question is whether we can use the sparse polynomial interpolation technique by Ben-Or and Tiwari [7] (see also [49, 24] for alternative descriptions of their approach) to give a more efficient deterministic algorithm for output-sensitive matrix multiplication. Indeed, they show how to use $\mathcal{O}(t)$ evaluations of a t-sparse polynomial p to efficiently interpolate p (for $p = g^{A,B}$, this corresponds to determining AB). Specifically, the $\mathcal{O}(t)$ evaluations define a certain Toeplitz system whose solution yields the coefficients of a polynomial $\zeta(Z) = \prod_{i=1}^z (Z - r_i)$ where r_i is the value of the i-th monomial of p evaluated at a certain known value. By factoring ζ into its linear factors, we can determine the monomials of p (i.e., for $p = g^{A,B}$, the nonzero entries of AB). In our case, we can then obtain AB by naive computations of the inner products at the nonzero positions in time $\mathcal{O}(nt)$. The bottleneck in this approach appears to be deterministic polynomial factorization into linear factors: In our setting, we would need to factor a degree-($\leq t$) polynomial over a prime field $\mathbb{F}_p$ of size $p = \Theta(n^2)$. We are not aware of deterministic algorithms faster than Shoup's $\mathcal{O}(t^{2+\varepsilon} \cdot \sqrt{p} \log^2 p)$-time algorithm [37], which would yield an $\mathcal{O}(n^2 + nt^{2+\varepsilon})$-time algorithm at best. However, such an algorithm would be dominated by Kutzkov's algorithm [28]. Can we sidestep this bottleneck? Note that some works improve on Shoup's running time for suitable primes (assuming the Extended Riemann Hypothesis; see [44, Chapter 14] for references).

References

1 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proc. 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*, pages 1681–1697, 2015. `doi:10.1137/1.9781611973730.112`.

2 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proc. 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS'14)*, pages 434–443, 2014. `doi:10.1109/FOCS.2014.53`.

3 Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proc. 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*, pages 218–230, 2015. `doi:10.1137/1.9781611973730.17`.

4 Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, 1997. `doi:10.1006/jcss.1997.1388`.

5 Rasmus Resen Amossen and Rasmus Pagh. Faster join-projects and sparse matrix multiplications. In *Proc. 12th International Conference on Database Theory (ICDT'09)*, pages 121–126, 2009. `doi:10.1145/1514894.1514909`.

6 Ilya Baran, Erik D. Demaine, and Mihai Patrascu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008. `doi:10.1007/s00453-007-9036-3`.

7 Michael Ben-Or and Prasoon Tiwari. A deterministic algorithm for sparse multivariate polynominal interpolation (extended abstract). In *Proc. 20th Annual ACM Symposium on Theory of Computing (STOC'88)*, pages 301–309, 1988. `doi:10.1145/62212.62241`.

8 Markus Bläser. Fast matrix multiplication. *Theory of Computing, Graduate Surveys*, 5:1–60, 2013. `doi:10.4086/toc.gs.2013.005`.

9 Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly sub-cubic algorithms for language edit distance and RNA-folding via fast bounded-difference min-plus product. In *Proc. 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS'16)*, pages 375–384, 2016. `doi:10.1109/FOCS.2016.48`.

10 Harry Buhrman and Robert Spalek. Quantum verification of matrix products. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06)*, pages 880–889, 2006. URL: `http://dl.acm.org/citation.cfm?id=1109557.1109654`.

11 Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the Strong Exponential Time Hypothesis and consequences for non-reducibility. In *Proc. 2016 ACM Conference on Innovations in Theoretical Computer Science (ITCS'16)*, pages 261–270, 2016. `doi:10.1145/2840728.2840746`.

12 Timothy M. Chan. More logarithmic-factor speedups for 3SUM, (median, +)-convolution, and some geometric 3SUM-hard problems. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, pages 881–897, 2018. `doi:10.1137/1.9781611975031.57`.

13 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal on Symbolic Computation*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.

14 Charles M. Fiduccia. Polynomial evaluation via the division algorithm: The fast Fourier transform revisited. In *Proc. 4th Annual ACM Symposium on Theory of Computing (STOC'72)*, pages 88–93, 1972. doi:10.1145/800152.804900.

15 Rusins Freivalds. Fast probabilistic algorithms. In *Proc. 8th International Symposium on Mathematical Foundations of Computer Science (MFCS'79)*, pages 57–69, 1979. doi:10.1007/3-540-09526-8_5.

16 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry*, 5:165–185, 1995. doi:10.1016/0925-7721(95)00022-2.

17 Leszek Gasieniec, Christos Levcopoulos, Andrzej Lingas, Rasmus Pagh, and Takeshi Tokuyama. Efficiently correcting matrix products. *Algorithmica*, 79(2):428–443, 2017. doi:10.1007/s00453-016-0202-3.

18 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. In *Proc. 55th IEEE Annual Symposium on Foundations of Computer Science (FOCS'14)*, pages 621–630, 2014. doi:10.1109/FOCS.2014.72.

19 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

20 Mark A. Iwen and Craig V. Spencer. A note on compressed sensing and the complexity of matrix multiplication. *Information Processing Letters*, 109(10):468–471, 2009. doi:10.1016/j.ipl.2009.01.010.

21 Riko Jacob and Morten Stöckel. Fast output-sensitive matrix multiplication. In *Proc. 23rd Annual European Symposium on Algorithms (ESA'15)*, pages 766–778, 2015. doi:10.1007/978-3-662-48350-3_64.

22 Zahra Jafargholi and Emanuele Viola. 3SUM, 3XOR, triangles. *Algorithmica*, 74(1):326–343, 2016. doi:10.1007/s00453-014-9946-9.

23 Stacey Jeffery, Robin Kothari, and Frédéric Magniez. Improving quantum query complexity of boolean matrix multiplication using graph collision. In *Proc. 39th International Colloquium on Automata, Languages, and Programming (ICALP'12)*, pages 522–532, 2012. doi:10.1007/978-3-642-31594-7_44.

24 Erich Kaltofen and Yagati N. Lakshman. Improved sparse multivariate polynomial interpolation algorithms. In *Proc. 1st International Symposium on Symbolic and Algebraic Computation (ISSAC'88)*, pages 467–474, 1988. doi:10.1007/3-540-51084-2_44.

25 Daniel M. Kane, Shachar Lovett, and Shay Moran. Near-optimal linear decision trees for k-SUM and related problems. *CoRR*, abs/1705.01720, 2017. To appear in STOC'18. arXiv:1705.01720.

26 Tracy Kimbrel and Rakesh K. Sinha. A probabilistic algorithm for verifying matrix products using $O(n^2)$ time and $\log_2 n + O(1)$ random bits. *Information Processing Letters*, 45(2):107–110, 1993. doi:10.1016/0020-0190(93)90224-W.

27 Marvin Künnemann. On nondeterministic derandomization of Freivalds' algorithm: Consequences, avenues and algorithmic progress. *CoRR*, abs/1806.09189, 2018. arXiv:1806.09189.

28 Konstantin Kutzkov. Deterministic algorithms for skewed matrix products. In *Proc. 30th International Symposium on Theoretical Aspects of Computer Science (STACS'13)*, pages 466–477, 2013. doi:10.4230/LIPIcs.STACS.2013.466.

29 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. 39th International Symposium on Symbolic and Algebraic Computation (ISSAC'14)*, pages 296–303, 2014. doi:10.1145/2608628.2608664.

30 Andrzej Lingas. A fast output-sensitive algorithm for boolean matrix multiplication. In *Proc. 17th Annual European Symposium on Algorithms (ESA'09)*, pages 408–419, 2009. `doi:10.1007/978-3-642-04128-0_37`.

31 Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, 2011. `doi:10.1016/j.cosrev.2010.09.009`.

32 Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 22(4):838–856, 1993. `doi:10.1137/0222053`.

33 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 026(2):415–419, 1985. URL: `http://eudml.org/doc/17394`.

34 Rasmus Pagh. Compressed matrix multiplication. *ACM Transactions on Computation Theory*, 5(3):9:1–9:17, 2013. `doi:10.1145/2493252.2493254`.

35 Daniel S. Roche. Error correction in fast matrix multiplication and inverse. *CoRR*, abs/1802.02270, 2018. `arXiv:1802.02270`.

36 Claus-Peter Schnorr and C. R. Subramanian. Almost optimal (on the average) combinatorial algorithms for boolean matrix product witnesses, computing the diameter (extended abstract). In *Proc. 2nd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM'98)*, pages 218–231, 1998. `doi:10.1007/3-540-49543-6_18`.

37 Victor Shoup. On the deterministic complexity of factoring polynomials over finite fields. *Information Processing Letters*, 33(5):261–267, 1990. `doi:10.1016/0020-0190(90)90195-4`.

38 Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, Aug 1969. `doi:10.1007/BF02165411`.

39 Terence Tao, Ernest Croot III, and Harald Helfgott. Deterministic methods to find primes. *Mathematics of Computation*, 81(278):1233–1246, 2012. `doi:10.1090/S0025-5718-2011-02542-1`.

40 Leslie G. Valiant. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10(2):308–315, 1975. `doi:10.1016/S0022-0000(75)80046-8`.

41 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. 44th Annual ACM Symposium on Theory of Computing Conference (STOC'12)*, pages 887–898, 2012. `doi:10.1145/2213977.2214056`.

42 Virginia Vassilevska Williams. Fine-grained algorithms and complexity. In *Proc. 21st International Conference on Database Theory (ICDT'18)*, pages 1:1–1:1, 2018. `doi:10.4230/LIPIcs.ICDT.2018.1`.

43 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proc. 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*, pages 645–654, 2010. `doi:10.1109/FOCS.2010.67`.

44 Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra (3. ed.)*. Cambridge University Press, 2013.

45 Jirí Wiedermann. Fast nondeterministic matrix multiplication via derandomization of Freivalds' algorithm. In *Proc. 8th IFIP International Conference on Theoretical Computer Science (TCS'14)*, pages 123–135, 2014. `doi:10.1007/978-3-662-44602-7_11`.

46 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proc. 46th Annual ACM Symposium on Theory of Computing (STOC'14)*, pages 664–673, 2014. `doi:10.1145/2591796.2591811`.

47 Ryan Williams. Strong ETH breaks with Merlin and Arthur: Short non-interactive proofs of batch evaluation. In *Proc. 31st Conference on Computational Complexity (CCC'16)*, pages 2:1–2:17, 2016. `doi:10.4230/LIPIcs.CCC.2016.2`.

48 Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Transactions on Algorithms*, 1(1):2–13, 2005. doi:10.1145/1077464.1077466.

49 Richard Zippel. Interpolating polynomials from their values. *Journal of Symbolic Computation*, 9(3):375–403, 1990. doi:10.1016/S0747-7171(08)80018-1.

Optimal Online Contention Resolution Schemes via Ex-Ante Prophet Inequalities

Euiwoong Lee
Courant Institute of Mathematical Sciences, New York University, New York City, USA
euiwoong@cims.nyu.edu

Sahil Singla
Computer Science Department, Carnegie Mellon University, Pittsburgh, USA
ssingla@cmu.edu

Abstract

Online contention resolution schemes (OCRSs) were proposed by Feldman, Svensson, and Zenklusen [11] as a generic technique to round a fractional solution in the matroid polytope in an online fashion. It has found applications in several stochastic combinatorial problems where there is a *commitment* constraint: on seeing the value of a stochastic element, the algorithm has to immediately and irrevocably decide whether to select it while always maintaining an independent set in the matroid. Although OCRSs immediately lead to prophet inequalities, these prophet inequalities are not optimal. Can we instead use prophet inequalities to design optimal OCRSs?

We design the first optimal 1/2-OCRS for matroids by reducing the problem to designing a matroid prophet inequality where we compare to the stronger benchmark of an ex-ante relaxation. We also introduce and design optimal $(1-1/e)$-random order CRSs for matroids, which are similar to OCRSs but the arrival order is chosen uniformly at random.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Prophets, Contention Resolution, Stochastic Optimization, Matroids

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.57

Related Version A full version of the paper is available at `https://arxiv.org/abs/1806.09251`.

Acknowledgements We are thankful to Ravishankar Krishnaswamy and Deeparnab Chakrabarty for useful discussions in early part of this project. Part of this work was done while the authors were visiting the Simons Institute for the Theory of Computing. The second author was supported in part by NSF awards CCF-1319811, CCF-1536002, and CCF-1617790.

1 Introduction

Given a combinatorial optimization problem, a common algorithmic approach is to first solve a convex relaxation of the problem and to then round the obtained fractional solution $\mathbf{x}$ into a *feasible integral* solution while (approximately) preserving the objective. Contention resolution schemes (CRSs), introduced in [8], is a way to perform this rounding given a fractional solution $\mathbf{x} \in \mathbb{R}^n_{\geq 0}$. For $c > 0$, intuitively a c-CRS is a rounding algorithm that guarantees every element i is selected into the final feasible solution w.p. at least $c \cdot x_i$. For a *maximization* problem with a linear objective, by linearity of expectation such a c-CRS directly implies a c-approximation algorithm.

© Euiwoong Lee and Sahil Singla;
licensed under Creative Commons License CC-BY

In a recent work, Feldman et al. [11] introduced an Online CRS (OCRS), which is a CRS with an additional property that it performs the rounding in an "online fashion". This property is crucial for the *prophet inequality* problem (or any stochastic combinatorial problem with a *commitment* constraint; see §1.3).

► **Definition 1** (Prophet inequality)**.** Suppose each element $i \in N$ takes a value $v_i \in \mathbb{R}_{\geq 0}$ independently from some known distribution $\mathcal{D}_i$. These values are presented one-by-one to an online algorithm in an adversarial order. Given a packing feasibility constraint $\mathcal{F} \subseteq 2^N$, the problem is to immediately and irrevocably decide whether to select the next element i, while always maintaining a feasible solution and maximizing the sum of the selected values.

A *c-approximation prophet inequality* for $0 \leq c \leq 1$ means there exists an online algorithm with expected value at least c times the expected value of an offline algorithm that knows all values from the beginning. As shown in [11], a c-OCRS immediately implies a c-approximation prophet inequality. Some other applications are oblivious posted pricing mechanisms and stochastic probing.

Although powerful, the above approach of using OCRSs to design prophet inequalities does not give us *optimal* prophet inequalities. For example, while we know a 1/2-approximation prophet inequality over matroids [20], we only know a 1/4-OCRS over matroids [11]. This indicates that the currently known OCRSs may not be optimal. Can we design better OCRSs? The main contribution of this work is to design an *optimal* OCRS over matroid constraints using the following idea:

Not only can we design prophet inequalities from OCRSs, we can also design OCRSs from prophet inequalities.

More specifically, our OCRS is based on an *ex-ante* prophet inequality: we compare the online algorithm to the stronger benchmark of a convex relaxation. We modify existing prophet inequalities to obtain ex-ante prophet inequalities while *preserving* the approximation factors. As a corollary, this gives the first optimal 1/2-OCRS over matroids.

Since for many applications the arrival order is not chosen by an adversary, some recent works have also studied prophet secretary inequalities where the arrival order is chosen uniformly at random [10, 9, 5]. Motivated by these works, we introduce *random order contention resolution schemes* (RCRS), which is an OCRS for uniformly random arrival[1]. Again by designing the corresponding random order ex-ante prophet inequalities, we obtain optimal $(1 - 1/e)$-RCRS over matroids.

In §1.1 we formally define an OCRS/RCRS and an ex-ante prophet inequality. In §1.2 we describe our results and proof techniques. See §1.3 for further related work.

1.1 Model

CRSs are a powerful tool for offline and stochastic optimization problems [8, 15]. For a given $\mathbf{x} \in [0,1]^N$, let $R(\mathbf{x})$ denote a random set containing each element $i \in N$ independently w.p. x_i. We say an element i is *active* if it belongs to $R(\mathbf{x})$.

► **Definition 2** (Contention resolution scheme)**.** Given a finite ground set N with $n = |N|$ and a packing (downward-closed) family of feasible subsets $\mathcal{F} \subseteq 2^N$, let $P_{\mathcal{F}} \subseteq [0,1]^N$ be the convex hull of all characteristic vectors of feasible sets. For a given $\mathbf{x} \in P_{\mathcal{F}}$, a *c-selectable*

[1] A parallel independent work has also introduced RCRS [1]; however, their technical results are very different.

CRS (or simply, c-CRS) is a (randomized) mapping $\pi : 2^N \to 2^N$ satisfying the following three properties:

(i) $\pi(S) \subseteq S$ for all $S \subseteq N$.
(ii) $\pi(S) \in \mathcal{F}$ for all $S \subseteq N$.
(iii) $\Pr_{R(\mathbf{x}),\pi}[i \in \pi(R(\mathbf{x}))] \geq c \cdot x_i$ for all $i \in N$.

Notice, if f is a monotone linear function then $\mathbb{E}[f(\pi(R(\mathbf{x})))] \geq c \cdot \mathbb{E}[f(R(\mathbf{x}))]$. By constructing CRSs for various constraint families of $\mathcal{F}$, Chekuri et al. [8] give improved approximation algorithms for linear and submodular maximization problems under knapsack, matroid, matchoid constraints, and their intersections[2].

In the above applications to offline optimization problems, the algorithm first flips all the random coins to sample $R(\mathbf{x})$, and then obtains $\pi(R(\mathbf{x})) \subseteq R(\mathbf{x})$. For various online problems such as the prophet inequality, this randomness is an inherent part of the problem. Feldman et al. [11] therefore introduce an OCRS where the random set $R(\mathbf{x})$ is sampled in the same manner, but whether $i \in R(\mathbf{x})$ (or not) is only revealed one-by-one to the algorithm in an adversarial order[3]. After each revelation (arrival), the OCRS has to irrevocably decide whether to include $i \in R(\mathbf{x})$ into $\pi(R(\mathbf{x}))$ (if possible). A c-selectable OCRS (or simply, c-OCRS) is an OCRS satisfying the above properties (i) to (iii) of a c-CRS.

In this work, we also study RCRS which is an OCRS with the arrival order chosen uniformly at random. A c-selectable RCRS (or simply, c-RCRS) is an RCRS satisfying the above properties (i) to (iii) of a c-CRS, where in Property (iii) we also take expectation over the arrival order.

While prophet inequalities have been designed using OCRSs, our main result in this paper is to show a deeper *reverse* connection between OCRSs and prophet inequalities. We first define an ex-ante prophet inequality. Given a prophet inequality problem instance with packing constraints $\mathcal{F}$ and r.v.s $v_i \sim \mathcal{D}_i$ for $i \in N$, the following *ex-ante relaxation* gives an upper bound on the expected offline optimum:

$$\max_{\mathbf{x}} \sum_i x_i \cdot \mathbb{E}_{v_i \sim \mathcal{D}_i}[v_i \mid v_i \text{ takes value in its top } x_i \text{ quantile}] \quad \text{s.t.} \quad \mathbf{x} \in P_{\mathcal{F}}. \tag{1}$$

To prove that (1) is an upper bound, we interpret x_i as the probability that i is in the offline optimum. It is also known that (1) is a convex program and can be solved efficiently; see [11] for more details.

▶ **Definition 3** (Ex-ante prophet inequality)**.** For $0 \leq c \leq 1$, a c-approximation ex-ante prophet inequality for packing constraints $\mathcal{F}$ is a prophet inequality algorithm with expected value at least c times (1).

Before describing our results, to build some intuition for the above definitions we discuss the special case of a rank 1 matroid, i.e., where we can only select one of the n elements.

Example: Rank 1 matroid

For simplicity, in this section we assume that all random variables are Bernoulli, i.e., v_i takes value y_i independently w.p. p_i, and is 0 otherwise. We first show why a c-OCRS implies a c-approximation prophet inequality for rank 1 matroids.

[2] Some "greedy" properties are also required from the CRS for the guarantees to hold for a submodular function f [8].
[3] For adversarial arrival order, we assume that this order is known to the OCRS algorithm in advance. This offline adversary is weaker than the almighty adversary considered in [11], but is common in the prophet inequality literature [24, 25]. We need this assumption in §2 to define our exponential sized linear program.

Consider the optimum solution $\mathbf{x}$ to the ex-ante relaxation (1) for the above Bernoulli instance. Its objective value is $\sum_i x_i y_i$ where $\mathbf{x}$ satisfies $\sum_i x_i \leq 1$. Moreover, $x_i \leq p_i$ for all i because selecting i beyond p_i does not increase (1). To see why (1) gives an upper bound on the expected offline maximum, observe that if we interpret x_i as the probability that v_i is the offline maximum, this gives a feasible solution to $\sum_i x_i \leq 1$ and with value at most $\sum_i x_i y_i$. Thus, to prove a c-approximation prophet inequality, it suffices to design an online algorithm with value at least $c \cdot \sum_i x_i y_i$. Consider an algorithm that runs a c-OCRS on $\mathbf{x}$, where i is considered active independently w.p. x_i/p_i whenever v_i takes value y_i. This ensures element i is active w.p. exactly x_i. Since a c-OCRS guarantees each element is selected w.p. $\geq c$ when it is active, by linearity of expectation such an algorithm has expected value at least $c \cdot \sum_i x_i y_i$.

We now discuss a simple 1/4-OCRS for a rank 1 matroid. Given $\mathbf{x}$ satisfying $\sum_i x_i \leq 1$, consider an algorithm that ignores each element i independently w.p. 1/2, and otherwise selects i only if it is active. Since this algorithm selects any element i w.p. at most $x_i/2$ (when i is not ignored and is active), by Markov's inequality the algorithm selects no element till the end w.p. at least $1 - \sum_i x_i/2 \geq 1/2$. Hence the algorithm reaches each element i w.p. at least 1/2 without selecting any of the previous elements. Moreover, it does not ignore i w.p. 1/2, which implies it considers each element w.p. at least 1/4. The OCRS due to Feldman et al. [11] can be thought of generalizing this approach to a general matroid.

An interesting result of Alaei [4] shows that the above 1/4-OCRS can be improved to a 1/2-OCRS over a rank 1 matroid by "greedily" maximizing the probability of ignoring the next element i, but considering i w.p. 1/2 on average. In the full version of the paper we present Alaei's proof for completeness, and also show how to obtain a simple $(1 - 1/e)$-RCRS for a rank 1 matroid. This raises the question whether one can obtain a 1/2-OCRS and a $(1 - 1/e)$-RCRS for general matroids.

1.2 Results and Techniques

Our first theorem gives an approximation factor preserving reduction from OCRSs to ex-ante prophet inequalities.

▶ **Theorem 4.** *For $0 \leq c \leq 1$, a c-approximation ex-ante prophet inequality for adversarial (random) arrival order over a packing constraint $\mathcal{F}$ implies a c-OCRS (c-RCRS) over $\mathcal{F}$.*

We complement the above theorem by designing ex-ante prophet inequalities over matroids.

▶ **Theorem 5.** *For matroids, there exists a 1/2-approximation ex-ante prophet inequality for adversarial arrival order and a $(1 - 1/e)$-approximation ex-ante prophet inequality for uniformly random arrival order.*

As a corollary, the above two theorems give optimal OCRS and RCRS over matroids. This generalizes the rank 1 results discussed in the previous section to general matroids; although the proof techniques are very different.

▶ **Corollary 6.** *For matroids, there exists a 1/2-OCRS and a $(1 - 1/e)$-RCRS.*

Our 1/2-OCRS above assumes that the arrival order is known to the algorithm. It is an interesting open question to find a 1/2-OCRS for an almighty/online adversary as in [11].

We first prove that both the factors 1/2 and $(1 - 1/e)$ in Corollary 6 are *optimal.*

Optimality of 1/2-OCRS and $(1 - 1/e)$-RCRS

We argue that the factors 1/2 and $(1 - 1/e)$ in Corollary 6 are optimal even in the special case of a rank 1 matroid. For adversarial arrival, consider just two elements, i.e., $n = 2$, with

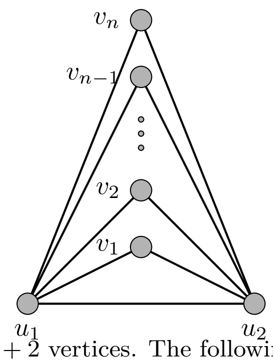


Figure 1 The Hat example on $n+2$ vertices. The following $\mathbf{x}$ belongs to the graphic matroid: $x_e = 1/2$ for $e = (u_i, v_j)$ where $i \in \{1,2\}$ and $j \in \{1,\dots,n\}$, and $x_e = 1$ for $e = (u_1, u_2)$.

$x_1 = 1-\epsilon$ and $x_2 = \epsilon$ for some $\epsilon \to 0$. Since the OCRS algorithm has to select the first element at least $1/2$ fraction of the times, it can attempt to select the second element at most $1/2 + \epsilon/2$ fraction of the times.

For random arrival order, consider the feasible solution $\mathbf{x}$ with $x_i = 1/n$ for every $i \in N$. We show that no online RCRS algorithm can guarantee each element is selected w.p. greater than $\frac{(1-1/e)}{n}$. This is because for the product distribution, w.p. $1/e$ none of the n elements is active (more precisely, w.p. $(1-1/n)^n$). Hence the RCRS algorithm, which only selects active elements, selects some element w.p. $1-1/e$. This implies on average it cannot pick every element w.p. greater than $\frac{(1-1/e)}{n}$. This example, originally shown in [8], also proves that offline CRS cannot better than $(1-1/e)$-selectable.

Our techniques

We first see the difficulty in extending Alaei's greedy approach from a rank 1 matroid to a general matroid. Consider the graphic matroid for the *Hat* example (see Figure 1). Suppose the base edge (u_1, u_2) appears in the end of an adversarial order. Notice that any algorithm which ignores the structure of the matroid is very likely to select some pair of edges (u_1, v_i) and (v_i, u_2) for some i. Since this pair spans the base edge (u_1, u_2), such an OCRS algorithm will not satisfy c-selectability for (u_1, u_2). To overcome this, Feldman et al. [11] decompose the matroid into "simpler" matroids using $\mathbf{x}$. However, it is not clear how to extend their approach beyond a 1/4-OCRS.

In this paper we take an alternate LP based approach to design OCRSs, which was first used by Chekuri et al. [8] to design offline CRSs. The idea is to define an exponential sized linear program where each variable denotes a deterministic OCRS algorithm. The objective of this linear program is to maximize c s.t. each element is selected at least c fraction of the times (c-selectability). Thus to show existence of a 1/2-OCRS, it suffices to prove this linear program has value $c \geq 1/2$. In §2 we prove this by showing that the dual LP has value at least $1/2$ because it can be interpreted as an ex-ante prophet inequality.

Next, to show there exists a $1/2$ approximation ex-ante prophet inequality, our approach is inspired from the matroid prophet inequality of Kleinberg and Weinberg [20]. They give an online algorithm that gets at least half of the expected offline optimum for the product distribution (independent r.v.s). Unfortunately, their techniques do not directly extend because the ex-ante relaxation objective could be significantly higher than for the product distribution (this is known as the *correlation gap*, which can be $e/(e-1)$ [2, 6]). Our primary technique is to view the ex-ante relaxation solution as a "special kind" of a correlated value distribution. Although prophet inequalities are not possible for general correlated distributions [19], we show that in this special case the original proof of the matroid prophet inequality algorithm retains its $1/2$ approximation after some modifications.

1.3 Further Related Work

Krengel and Sucheston gave the first tight 1/2-single item prophet inequality [22, 21]. The connection between multiple-choice prophet inequalities and mechanism design was recognized in [18]; they proved a prophet inequality for uniform matroids. This bound was later improved by Alaei [3] using the *Magician's problem*, which is an OCRS in disguise. Chawla et al. [7] further developed the connection between prophet inequalities and mechanism design, and showed how to be $O(1)$-prophet inequality for general matroids in a variant where the algorithm may choose the element order. Yan [26] improved this result to $e/(e-1)$-competitive using the *correlation gap* for submodular functions, first studied in [2, 6]. Chekuri et al. [8] adapted correlation gaps to a polytope to design CRSs. Improved correlation gaps were presented in [26, 17]. The matroid prophet inequality was first explicitly formulated in [20]. Feldman et al. [11] gave an alternate proof, and extended to Bernoulli submodular functions, using OCRSs. Finally, information theoretic $O(\text{poly}\log(n))$-prophet inequalities are also known for general downward-closed constraints [24, 25].

The prophet secretary notion was first introduced in [10], where the elements arrive in a uniformly random order and draw their values from known independent distributions. Their results have been recently improved [9, 5]. There is a long line of work on studying the *commitment constraints* for combinatorial probing problems, e.g., see [13, 15, 16, 14]. In these models the algorithm starts with some stochastic knowledge about the input and on probing an element has to irrevocably commit if the element is to be included in the final solution. A common approach to handle such a constraint is using a prophet inequality/OCRS.

2 OCRS Assuming an Ex-Ante Prophet Inequality

In this section we prove Theorem 4, showing how to reduce the problem of designing an OCRS to a prophet inequality where we compare ourself to the ex-ante relaxation instead of the expected offline maximum.

2.1 Using LP Duality

Given a finite ground set N with $n = |N|$ and a downward-closed family of feasible subsets $\mathcal{F} \subseteq 2^N$, let $P_{\mathcal{F}} \subseteq [0,1]^N$ be the convex hull of all characteristic vectors of feasible sets. Let $\mathbf{x} \in P_{\mathcal{F}}$ and $R(\mathbf{x})$ denote a random set containing each element $i \in N$ independently w.p. x_i. For offline CRSs, let Φ^* be the set of valid offline *deterministic* mappings; i.e., $\phi : 2^N \to \mathcal{F}$ is in Φ^* iff $\phi(A) \subseteq A$ and $\phi(A) \in \mathcal{F}$ for all $A \subseteq N$. For $\phi \in \Phi^*$ and $i \in N$, let $q_{i,\phi} := \Pr_{R(\mathbf{x})}[i \in \phi(R(\mathbf{x}))]$ denote the probability of selecting i if the CRS executes ϕ. The following LP relaxation, introduced by Chekuri et al. [8], finds a c-selectable *randomized* CRS. It has variables $\{\lambda_\phi\}_{\phi\in\Phi^*}$ and c.

$$\begin{aligned}
\max_{\boldsymbol{\lambda},c}\quad & c \\
\text{s.t.}\quad & \sum_{\phi\in\Phi^*} q_{i,\phi}\lambda_\phi \geq x_i \cdot c && i \in N \\
& \sum_{\phi\in\Phi^*} \lambda_\phi = 1 \\
& \lambda_\phi \geq 0 && \forall \phi \in \Phi^*
\end{aligned}$$

Observe that if the above LP has value c, there exists a randomized c-CRS. This is because we can randomly select one of the ϕ's w.p. λ_ϕ, and the constraint $\sum_{\phi\in\Phi^*} q_{i,\phi}\lambda_\phi \geq x_i \cdot c$

ensures c-selectability for every $i \in N$. Chekuri et al. noticed that by strong duality, to prove the above LP has value at least c, it suffices to show that the following dual program has value at least c. It has variables $\{y_i\}_{i\in N}$ and μ.

$$\begin{aligned}
&\min_{\boldsymbol{y},\mu} \ \mu \\
&\quad \text{s.t.} \ \sum_{i\in N} q_{i,\phi} y_i \le \mu && \phi \in \Phi^* \\
&\qquad\quad \sum_{i\in N} x_i y_i = 1 \\
&\qquad\quad y_i \ge 0 && \forall i \in N
\end{aligned}$$

To design OCRSs (RCRSs), we take a similar approach as Chekuri et al and let Φ^* be the set of all *deterministic online algorithms*. Formally, $\phi : 2^N \times 2^N \times N \to \{0,1\}$ belongs to Φ^* iff $\phi(A,B,i) = 1$ only for $B \subseteq A$, $i \notin A$, and $B \cup \{i\} \in \mathcal{F}$. Intuitively, $\phi(A,B,i) = 1$ indicates that the online algorithm selects element i in the current iteration after processing elements in A and selecting elements in B. Let $q_{i,\phi}$ denote the probability of selecting i if the OCRS (RCRS) executes ϕ, where for RCRS we also take probability over the random order. By the above duality argument, to show existence of a c-OCRS (c-RCRS) it suffices to prove the dual LP has value at least c. We prove this by showing that for any $\boldsymbol{y} \ge 0$ s.t. $\sum_{i\in N} x_i y_i = 1$, there exists $\phi \in \Phi^*$ such that $\sum_{i\in N} q_{i,\phi} y_i \ge c$.

Consider a Bernoulli prophet inequality instance where each element $i \in N$ has value y_i with probability x_i, and 0 otherwise. Since $\mathbf{x} \in P_{\mathcal{F}}$, notice that $\sum_{i\in N} x_i y_i = 1$ is exactly the value of the ex-ante relaxation (1) for this instance. Thus, a c-approximation ex-ante prophet inequality implies there exists a $\phi \in \Phi^*$ with value at least c. By linearity of expectation, the value of ϕ is $\sum_{i\in N} q_{i,\phi} y_i$, which proves $\sum_{i\in N} q_{i,\phi} y_i \ge c$.

2.2 Solving the LP Efficiently

While the original primal LP has an exponential number of variables, we can compute an OCRS (or RCRS) that achieves value at least c as follows. In the dual program, given $\boldsymbol{y}$ s.t. $\sum_i x_i y_i = 1$, we can use the ex-ante prophet inequality to find $\phi \in \Phi^*$ with value $\sum_i q_{i,\phi} y_i \ge c$ in polynomial time. (Notice $q_{i,\phi}$ can be computed in polynomial time because the adversarial order is known to the OCRS algorithm.) This implies for any $\epsilon > 0$, the polytope $Q_{c-\epsilon} := \{\boldsymbol{y} : \boldsymbol{y} \ge 0, \sum_i x_i y_i = 1, \sum_i q_{i,\phi} y_i \le c - \epsilon \text{ for all } \phi \in \Phi^*\}$ is empty.

Since we have an efficient *separation oracle* (for any y, we can find a violated constraint in polynomial time) for $Q_{c-\epsilon}$, by running the ellipsoid algorithm [12] we can find a subset $\Phi' \subseteq \Phi^*$ with $|\Phi'| = \text{poly}(n)$ in polynomial time such that $Q'_{c-\epsilon} := \{\boldsymbol{y} : \boldsymbol{y} \ge 0, \sum_i x_i y_i = 1, \sum_i q_{i,\phi} y_i \le c - \epsilon \text{ for all } \phi \in \Phi'\}$ is empty. Now the following linear program, which has a polynomial number of variables and constraints, with optimal value at least $c - \epsilon$ can be solved efficiently.

$$\begin{aligned}
&\max_{\boldsymbol{\lambda},c} \ c \\
&\quad \text{s.t.} \ \sum_{\phi\in\Phi'} q_{i,\phi} \lambda_\phi \ge x_i \cdot c && i \in N \\
&\qquad\quad \sum_{\phi\in\Phi'} \lambda_\phi = 1 \\
&\qquad\quad \lambda_\phi \ge 0 && \forall \phi \in \Phi'
\end{aligned}$$

3 Ex-Ante Prophet Inequalities for a Matroid

This section proves Theorem 5 by designing for a matroid a 1/2-ex-ante prophet inequality under adversarial arrival and a $(1-1/e)$-ex-ante prophet inequality under random arrival.

3.1 Notation

Let $\mathbf{v} \sim \mathcal{D}$ be a set of random element values $\{v_1, \dots, v_n\}$ where each v_i is independently drawn from $\mathcal{D}_i$. Let $\mathbf{x}$ be the optimal solution to the ex-ante relaxation in (1) for a given matroid $\mathcal{M} = (N, \mathcal{I})$. For $i \in N$, denote

$$y_i := \mathbb{E}_{v_i \sim \mathcal{D}_i}[v_i \mid v_i \text{ takes value in its top } x_i \text{ quantile}]. \tag{2}$$

Since $\mathbf{x} \in P_{\mathcal{M}}$, we can write it as a convex combination of independent sets in the matroid. In particular, this gives a correlated distribution $\hat{\mathcal{D}}$ over independent sets of $\mathcal{M}$ such that for each $i \in N$, we have $\Pr_{I \sim \hat{\mathcal{D}}}[i \in I] = x_i$. Let $\hat{\mathbf{v}} = \{\hat{v}_1, \dots, \hat{v}_n\}$ be a set of random values obtained by sampling $I \sim \hat{\mathcal{D}}$ and setting $\hat{v}_i = y_i$ for $i \in I$, and $\hat{v}_i = 0$ otherwise. Notice the optimal value of (1) is $\sum_i x_i y_i$ and for each $i \in N$, we have $\mathbb{E}[\hat{v}_i] = x_i y_i$.

We need the following notation to describe our algorithms.

▶ **Definition 7.** For any vector $\hat{\mathbf{v}}$ denoting values of elements of N and any $A \subseteq N$, we define:

- Let $\mathsf{Opt}(\hat{\mathbf{v}} \mid A) \subseteq N \setminus A$ denote the maximum value independent set in the contracted matroid $\mathcal{M}/A$.
- Let $R(A, \hat{\mathbf{v}}) := \sum_{i \in \mathsf{Opt}(\hat{\mathbf{v}} \mid A)} \hat{v}_i$ denote the *remaining value* after selecting set A.

We next define a *base price* of for every element i.

▶ **Definition 8.** For $A \in \mathcal{I}$ denoting an independent set of elements accepted by our algorithm, we define

- Let $b_i(A, \hat{\mathbf{v}}) := R(A, \hat{\mathbf{v}}) - R(A \cup \{i\}, \hat{\mathbf{v}})$ denote a threshold for element i.
- Let $b_i(A) := \mathbb{E}_{\hat{\mathbf{v}} \sim \hat{\mathcal{D}}}[b_i(A, \hat{\mathbf{v}})]$ denote the *base price* for element i.

3.2 Reducing to Bernoulli Distributions

In this section we show that it suffices to only prove Theorem 5 for Bernoulli distributions.

▶ **Lemma 9.** *If there exists an α-approximation ex-ante prophet inequality for Bernoulli distributed independent random values then there exists an α-approximation ex-ante prophet inequality for general distributed independent random values.*

Proof. Given a prophet inequality instance where $\mathbf{v} \sim \mathcal{D}$ for a general distribution $\mathcal{D}$, consider a new Bernoulli prophet inequality instance $\mathbf{v}' \sim \mathcal{D}'$ where for each $i \in N$, r.v. $v'_i \sim \mathcal{D}'_i$ independently takes value y_i (defined in (2)) w.p. x_i, and is 0 otherwise. Since the optimal ex-ante fractional value for both the general and Bernoulli instance is the same, to prove this theorem we use an ex-ante prophet inequality for the Bernoulli instance to design an ex-ante prophet inequality for the general instance with the same expected value.

On arrival of an element i, consider an algorithm for the general distribution that treats i is active iff v_i takes value in its top x_i quantile. If active, the algorithm asks the ex-ante prophet inequality of the Bernoulli instance to decide whether to select i. We claim that the expected value of this algorithm is $\alpha \cdot \sum_i x_i y_i$, which will prove this theorem. The claim is true because for the above algorithm each element i is active independently w.p. exactly x_i, and conditioned on being active its expected value is exactly y_i. Thus by linearity of expectation, the expected value is the same as the Bernoulli instance, which is $\alpha \cdot \sum_i x_i y_i$. ◀

3.3 Adversarial Order

We prove the optimal ex-ante prophet inequality for a matroid under the adversarial arrival.

▶ **Theorem 10.** *For matroids, there exists a* $1/2$*-approximation ex-ante prophet inequality for adversarial arrival order.*

Given the notation and definitions in §3.1, the proof of Theorem 10 is similar to the proof of the matroid prophet inequality in [20].

By Lemma 9, we know it suffices to prove this theorem only for Bernoulli distributions. Consider $\mathbf{v} \sim \mathcal{D}$ as the input to our online algorithm, where v_i takes value y_i w.p. x_i and is 0 otherwise. Given $\mathbf{v}$, our algorithm is deterministic and let $A := A(\mathbf{v})$ denote the set of elements that it selects. Relabel the elements such that the arrival order of the elements is $1, \ldots, n$. Let $A_i = A \cap \{1, \ldots, i\}$.

Our algorithm selects the next element i iff both $v_i > \mathcal{T}_i := \alpha \cdot b_i(A_{i-1})$ and selecting i is feasible in $\mathcal{M}$, where $\alpha = \frac{1}{2}$. Thus, the total value of algorithm $\mathsf{Alg} := \sum_{i \in A} v_i = \mathsf{Revenue} + \mathsf{Utility}$, where

$$\mathsf{Revenue} := \textstyle\sum_{i \in A} \mathcal{T}_i \qquad \text{and} \qquad \mathsf{Utility} := \textstyle\sum_{i \in A} (v_i - \mathcal{T}_i)^+.$$

Since $\sum_{i \in N} x_i y_i$ is the optimal value of (1), to prove Theorem 10 it suffices to show $\mathbb{E}[\mathsf{Alg}] = \mathbb{E}[\mathsf{Revenue}] + \mathbb{E}[\mathsf{Utility}] \geq \alpha \cdot \sum_{i \in N} x_i y_i$.

We keep track of the algorithm's progress using the following *residual* function:

$$r(i) := \mathbb{E}_{\mathbf{v} \sim \mathcal{D}, \hat{\mathbf{v}} \sim \hat{\mathcal{D}}}[R(A_{i-1}, \hat{\mathbf{v}})].$$

Clearly, $r(0) = \sum_{i \in N} x_i y_i$. In the following Lemma 11 and Lemma 12, we use the residual function to lower bound $\mathbb{E}[\mathsf{Revenue}]$ and $\mathbb{E}[\mathsf{Utility}]$.

▶ **Lemma 11.** $\mathbb{E}_{v \sim \mathcal{D}}[\mathsf{Revenue}] = \alpha \cdot \Big(r(0) - r(n)\Big)$.

Proof. From the definition of $\mathsf{Revenue}$, we get

$$\begin{aligned}\mathsf{Revenue} = \alpha \cdot \sum_{i \in A} b_i(A_{i-1}) \;&=\; \alpha \cdot \sum_{i \in A} \Big(\mathbb{E}_{\hat{\mathbf{v}}}[R(A_{i-1}, \hat{\mathbf{v}})] - \mathbb{E}_{\hat{\mathbf{v}}}[R(A_{i-1} \cup \{i\}, \hat{\mathbf{v}})]\Big) \\ &= \alpha \cdot \sum_{i \in A} \Big(\mathbb{E}_{\hat{\mathbf{v}}}[R(A_{i-1}, \hat{\mathbf{v}})] - \mathbb{E}_{\hat{\mathbf{v}}}[R(A_i, \hat{\mathbf{v}})]\Big) \\ &= \alpha \cdot \Big(\mathbb{E}_{\hat{\mathbf{v}}}[R(A_0, \hat{\mathbf{v}})] - \mathbb{E}_{\hat{\mathbf{v}}}[R(A, \hat{\mathbf{v}})]\Big).\end{aligned}$$

Taking expectation over $\mathbf{v} \sim \mathcal{D}$ and using definitions of $r(0)$ and $r(n)$, the lemma follows. ◀

▶ **Lemma 12.** $E_{v \sim \mathcal{D}}[\mathsf{Utility}] \geq (1 - \alpha) \cdot r(n)$.

Proof. We prove the following two inequalities:

$$\mathbb{E}_{\mathbf{v} \sim \mathcal{D}}[\mathsf{Utility}] \quad \geq \quad \mathbb{E}_{\mathbf{v} \sim \mathcal{D}, \hat{\mathbf{v}} \sim \hat{\mathcal{D}}}\Big[\sum_{i \in \mathsf{Opt}(\hat{\mathbf{v}}|A)} (\hat{v}_i - \mathcal{T}_i)^+\Big] \tag{3}$$

and

$$\mathbb{E}_{\mathbf{v} \sim \mathcal{D}, \hat{\mathbf{v}} \sim \hat{\mathcal{D}}}\Big[\sum_{i \in \mathsf{Opt}(\hat{\mathbf{v}}|A)} (\hat{v}_i - \mathcal{T}_i)^+\Big] \quad \geq \quad (1 - \alpha) \cdot \mathbb{E}_{\mathbf{v} \sim \mathcal{D}, \hat{\mathbf{v}} \sim \hat{\mathcal{D}}}[R(A, \hat{\mathbf{v}})]. \tag{4}$$

Lemma 12 now follows by summing (3) and (4), and using $r(n) = \mathbb{E}_{\mathbf{v} \sim \mathcal{D}, \hat{\mathbf{v}} \sim \hat{\mathcal{D}}}[R(A, \hat{\mathbf{v}})]$.

To prove (3), notice that for any i not selected by the algorithm, $v_i \le \mathcal{T}_i$. This implies

$$\mathbb{E}_{\mathbf{v}\sim\mathcal{D}}[\mathsf{Utility}] = \mathbb{E}_{\mathbf{v}}\Big[\sum_{i\in A}(v_i - \mathcal{T}_i)^+\Big] = \mathbb{E}_{\mathbf{v}}\Big[\sum_{i\in N}(v_i - \mathcal{T}_i)^+\Big].$$

Now observe that for any fixed i and $v_1,\ldots,v_{i-1}$, the threshold $\mathcal{T}_i$ is determined. Since v_i and $\hat{v}_i$ are independent random variables with the same distribution, we get

$$\mathbb{E}_{\mathbf{v}}[(v_i - \mathcal{T}_i)^+ | v_1,\ldots,v_{i-1}] = \mathbb{E}_{\mathbf{v},\hat{\mathbf{v}}}[(\hat{v}_i - \mathcal{T}_i)^+ | v_1,\ldots,v_{i-1}].$$

This implies

$$\mathbb{E}_{\mathbf{v}\sim\mathcal{D}}[\mathsf{Utility}] = \mathbb{E}_{\mathbf{v}}\Big[\sum_{i\in N}(v_i - \mathcal{T}_i)^+\Big] = \mathbb{E}_{\mathbf{v},\hat{\mathbf{v}}}\Big[\sum_{i\in N}(\hat{v}_i - \mathcal{T}_i)^+\Big] \ge \mathbb{E}_{\mathbf{v},\hat{\mathbf{v}}}\Big[\sum_{i\in \mathsf{Opt}(\hat{\mathbf{v}}|A)}(\hat{v}_i - \mathcal{T}_i)^+\Big].$$

Finally, to prove (4), we have

$$\begin{aligned}\mathbb{E}_{\mathbf{v},\hat{\mathbf{v}}}[R(A,\hat{\mathbf{v}})] = \mathbb{E}_{\mathbf{v},\hat{\mathbf{v}}}\Big[\sum_{i\in \mathsf{Opt}(\hat{\mathbf{v}}|A)}\hat{v}_i\Big] &\le \mathbb{E}_{\mathbf{v},\hat{\mathbf{v}}}\Big[\sum_{i\in \mathsf{Opt}(\hat{\mathbf{v}}|A)}\mathcal{T}_i\Big] + \mathbb{E}_{\mathbf{v},\hat{\mathbf{v}}}\Big[\sum_{i\in \mathsf{Opt}(\hat{\mathbf{v}}|A)}(\hat{v}_i - \mathcal{T}_i)^+\Big]\\ &\le \alpha\cdot\mathbb{E}_{\mathbf{v},\hat{\mathbf{v}}}[R(A,\hat{\mathbf{v}})] + \mathbb{E}_{\mathbf{v},\hat{\mathbf{v}}}\Big[\sum_{i\in \mathsf{Opt}(\hat{\mathbf{v}}|A)}(\hat{v}_i - \mathcal{T}_i)^+\Big],\end{aligned}$$

where the first inequality uses $\hat{v}_i \le \mathcal{T}_i + (\hat{v}_i - \mathcal{T}_i)^+$ and the second inequality uses Claim 13 for $S = \mathsf{Opt}(\hat{\mathbf{v}} \mid A)$. After rearranging, this implies (4). ◀

We need the following Claim 13 in the proof of Lemma 12.

▶ **Claim 13.** *For every pair of disjoint sets A, S such that $A\cup S \in \mathcal{M}$,*

$$\alpha\cdot\mathbb{E}_{\hat{\boldsymbol{v}}\sim\hat{\mathcal{D}}}\Big[\sum_{i\in S}R(A_{i-1},\hat{\boldsymbol{v}}) - R(A_{i-1}\cup\{i\},\hat{\boldsymbol{v}})\Big] = \sum_{i\in S}\mathcal{T}_i \le \alpha\cdot\mathbb{E}_{\hat{\boldsymbol{v}}\sim\hat{\mathcal{D}}}[R(A,\hat{\boldsymbol{v}})]. \qquad (5)$$

Proof. This directly follows from [20], as they proved it for every fixed $\hat{\mathbf{v}}$. The proof is similar to Claim 18 in the next section. ◀

Proof of Theorem 10. Using Lemma 11 and Lemma 12, and substituting $\alpha = \frac{1}{2}$, we get

$$\mathbb{E}[\mathsf{Alg}] = \mathbb{E}[\mathsf{Utility}] + \mathbb{E}[\mathsf{Revenue}] \ge \frac{1}{2}\cdot r(0) = \frac{1}{2}\cdot\sum_{i\in N}x_i y_i. \qquad ◀$$

3.4 Random Order

We prove the optimal ex-ante prophet inequality for a matroid for random arrival.

▶ **Theorem 14.** *For matroids, there exists a $(1-1/e)$-approximation ex-ante prophet inequality for uniformly random arrival order.*

The proof of Theorem 14 is similar to the matroid prophet secretary inequality in [9]. We consider the model where each item chooses the arrival time from $[0,1]$ uniformly and independently, which is equivalent to the random permutation model. Starting with $A_0 = \emptyset$, let A_t denote the set of accepted elements by our algorithm *before* time t. This is a random variable that depends on the values $\mathbf{v}$ and arrival times $\mathbf{T}$. For $t\in[0,1]$, let

$$\alpha(t) := 1 - \exp(t-1).$$

Suppose an element i arrives at time t, then our algorithm selects i iff both $v_i > \alpha(t) \cdot b_i(A_t)$ and selecting i is feasible in $\mathcal{M}$.

Similar to §3.3, we keep track of the algorithm's progress using the *residual* function

$$r(t) := \mathbb{E}_{\mathbf{v} \sim \mathcal{D}, \hat{\mathbf{v}} \sim \hat{\mathcal{D}}, \mathbf{T}}[R(A_t, \hat{\mathbf{v}})],$$

where A_t is a function of $\mathbf{v}$ and $\mathbf{T}$. Clearly, $r(0) = \sum_{i \in N} x_i y_i$.

▶ **Claim 15.** $\mathbb{E}_{v \sim \mathcal{D}, T}[\mathsf{Revenue}] = -\int_{t=0}^{1} \alpha(t) \cdot r'(t) dt.$

Proof. This follows directly from the definition of Revenue. See [9] for details. ◀

▶ **Lemma 16.** $\mathbb{E}_{v \sim \mathcal{D}, T}[\mathsf{Utility}] \geq \int_{t=0}^{1} (1 - \alpha(t)) \cdot r(t) dt.$

Proof. The utility for element i arriving at time t is given by

$$\mathbb{E}_{\mathbf{v},\mathbf{T}}[u_i \mid T_i = t] = \mathbb{E}_{\mathbf{v},\mathbf{T}_{-i}}\left[(v_i - \alpha(t) \cdot b_i(A_t))^+ \cdot \mathbf{1}_{i \notin \mathsf{Span}(A_t)} \;\middle|\; T_i = t\right].$$

Observe that A_t does not depend on v_i if $T_i = t$ because it includes only the acceptances *before* t. It does not depend on $\hat{v}_i$ either, as $\hat{v}_i$ is only used for analysis purposes and not known to the algorithm. Since v_i and $\hat{v}_i$ are identically distributed, we can also write

$$\mathbb{E}_{\mathbf{v} \sim \mathcal{D},\mathbf{T}}[u_i \mid T_i = t] = \mathbb{E}_{\mathbf{v} \sim \mathcal{D}, \hat{\mathbf{v}} \sim \hat{\mathcal{D}},\mathbf{T}_{-i}}\left[(\hat{v}_i - \alpha(t) \cdot b_i(A_t))^+ \cdot \mathbf{1}_{i \notin \mathsf{Span}(A_t)} \;\middle|\; T_i = t\right]. \tag{6}$$

Now observe that element i can belong to $\mathsf{Opt}(\hat{\mathbf{v}} \mid A_t)$ only if it's not already in $\mathsf{Span}(A_t)$, which implies $\mathbf{1}_{i \notin \mathsf{Span}(A_t)} \geq \mathbf{1}_{i \in \mathsf{Opt}(\hat{\mathbf{v}} \mid A_t)}$. Using this and removing non-negativity, we get

$$\mathbb{E}_{\mathbf{v},\mathbf{T}}[u_i \mid T_i = t] \geq \mathbb{E}_{\mathbf{v},\hat{\mathbf{v}},\mathbf{T}_{-i}}\left[(\hat{v}_i - \alpha(t) \cdot b_i(A_t)) \cdot \mathbf{1}_{i \in \mathsf{Opt}(\hat{\mathbf{v}} \mid A_t)} \;\middle|\; T_i = t\right].$$

Now we use Lemma 17 to remove the conditioning on element i arriving at time t as this gives a valid lower bound on expected utility,

$$\mathbb{E}_{\mathbf{v},\mathbf{T}}[u_i \mid T_i = t] \geq \mathbb{E}_{\mathbf{v},\hat{\mathbf{v}},\mathbf{T}}\left[(\hat{v}_i - \alpha(t) \cdot b_i(A_t)) \cdot \mathbf{1}_{i \in \mathsf{Opt}(\hat{\mathbf{v}} \mid A_t)}\right]. \tag{7}$$

We can now lower bound sum of all the utilities using Eq. (7) to get

$$\begin{aligned}
\mathbb{E}_{\mathbf{v},\mathbf{T}}[\mathsf{Utility}] &= \sum_i \int_{t=0}^{1} \mathbb{E}_{\mathbf{v},\mathbf{T}}[u_i \mid T_i = t] \cdot dt \\
&\geq \sum_i \int_{t=0}^{1} \mathbb{E}_{\mathbf{v},\hat{\mathbf{v}} \sim \hat{\mathcal{D}},\mathbf{T}}\left[(\hat{v}_i - \alpha(t) \cdot b_i(A_t)) \cdot \mathbf{1}_{i \in \mathsf{Opt}(\hat{\mathbf{v}} \mid A_t)}\right] \cdot dt.
\end{aligned}$$

By moving the sum over elements inside the integrals, we get

$$\begin{aligned}
\mathbb{E}_{\mathbf{v},\mathbf{T}}[\mathsf{Utility}] &\geq \int_{t=0}^{1} \mathbb{E}_{\mathbf{v},\hat{\mathbf{v}},\mathbf{T}}\left[\sum_i (\hat{v}_i - \alpha(t) \cdot b_i(A_t)) \cdot \mathbf{1}_{i \in \mathsf{Opt}(\hat{\mathbf{v}} \mid A_t)}\right] \cdot dt \\
&= \int_{t=0}^{1} \mathbb{E}_{\mathbf{v},\hat{\mathbf{v}},\mathbf{T}}\left[R(A_t, \hat{\mathbf{v}}) - \alpha(t) \cdot \sum_{i \in \mathsf{Opt}(\hat{\mathbf{v}} \mid A_t)} b_i(A_t)\right] \cdot dt.
\end{aligned}$$

Finally, using Claim 18 for $S = \mathsf{Opt}(\hat{\mathbf{v}} \mid A_t)$, we get

$$\mathbb{E}_{\mathbf{v},\mathbf{T}}[\mathsf{Utility}] \geq \int_{t=0}^{1} \mathbb{E}_{\mathbf{v},\hat{\mathbf{v}},\mathbf{T}}\left[(1 - \alpha(t)) \cdot R(A_t, \hat{\mathbf{v}})\right] \cdot dt.$$ ◀

Proof of Theorem 14. Using Lemma 16 and Claim 15, we get

$$\begin{aligned}\mathbb{E}[\mathsf{Alg}] &= \mathbb{E}[\mathsf{Revenue}] + \mathbb{E}[\mathsf{Utility}] \\ &\geq -\int_{t=0}^{1} \alpha(t)\cdot r'(t)\cdot dt + \int_{t=0}^{1} (1-\alpha(t))\cdot r(t)\cdot dt \\ &= \int_{t=0}^{1} r(t)\cdot(1-\alpha(t)+\alpha'(t))\cdot dt - [r(t)\cdot\alpha(t)]_{t=0}^{1}.\end{aligned}$$

Notice that for $\alpha(t) = 1 - e^{t-1}$, we have $1 - \alpha(t) + \alpha'(t) = 0$. Hence, we get

$$\mathbb{E}[\mathsf{Alg}] \geq -[r(t)\cdot\alpha(t)]_{t=0}^{1} = \left(1-\frac{1}{e}\right)\cdot r(0) = \left(1-\frac{1}{e}\right)\cdot\sum_{i\in N} x_i y_i. \qquad \blacktriangleleft$$

Finally, we prove the missing Lemma 17 that removes the conditioning on i arriving at t.

▶ **Lemma 17.** *For any i, any time t, and any fixed $\boldsymbol{v}, \hat{\boldsymbol{v}}$, we have*

$$\mathbb{E}_{\boldsymbol{T}_{-i}}\left[(\hat{v}_i - \alpha(t)\cdot b_i(A_t))\cdot \mathbf{1}_{i\in\mathsf{Opt}(\hat{\boldsymbol{v}}|A_t)} \mid T_i = t\right] \geq \mathbb{E}_{\boldsymbol{T}}\left[(\hat{v}_i - \alpha(t)\cdot b_i(A_t))\cdot \mathbf{1}_{i\in\mathsf{Opt}(\hat{\boldsymbol{v}}|A_t)}\right].$$

Proof. We prove the lemma for any fixed $\mathbf{T}_{-i}$. Suppose we draw a uniformly random $T_i \in [0,1]$. Observe that if $T_i \geq t$ then we have equality in the above equation because set A_t is the same both with and without i. This is also the case when $T_i < t$ but i is not selected into A_t. Finally, when $T_i < t$ and $i \in A_t$ we have $\mathbf{1}_{i\in\mathsf{Opt}(\hat{\mathbf{v}}|A_t)} = 0$ in the presence of element i (i.e., RHS of lemma), making the inequality trivially true. ◀

▶ **Claim 18.** *For any fixed $\boldsymbol{v}, \boldsymbol{T}$, time t, and set of elements $S \subseteq N$ that is independent in the matroid $\mathcal{M}/A_t$, we have*

$$\sum_{i\in S} b_i(A_t) \leq \mathbb{E}_{\hat{\boldsymbol{v}}}\left[R(A_t, \hat{\boldsymbol{v}})\right].$$

Proof. By definition

$$\textstyle\sum_{i\in S} b_i(A_t) = \mathbb{E}_{\hat{\mathbf{v}}}\left[\sum_{i\in S}\left(R(A_t,\hat{\mathbf{v}}) - R(A_t\cup\{i\},\hat{\mathbf{v}})\right)\right].$$

Fix the values $\hat{\mathbf{v}}$ arbitrarily, we also have

$$\sum_{i\in S}\left(R(A_t,\hat{\mathbf{v}}) - R(A_t\cup\{i\},\hat{\mathbf{v}})\right) \leq R(A_t,\hat{\mathbf{v}}).$$

This follows from the fact that $R(A_t,\hat{\mathbf{v}}) - R(A_t\cup\{i\},\hat{\mathbf{v}})$ are the respective critical values of the greedy algorithm on $\mathcal{M}/A_t$ with values $\hat{\mathbf{v}}$. Therefore, the bound follows from Lemma 3.2 in [23]. An alternative proof is given as Proposition 2 in [20] while in our case the first inequality can be skipped and the remaining steps can be followed replacing A by A_t.

Taking the expectation over $\hat{\mathbf{v}}$, the claim follows. ◀

References

1 Marek Adamczyk and Michal Wlodarczyk. Random order contention resolution schemes. *CoRR*, abs/1804.02584, 2018.

2 Shipra Agrawal, Yichuan Ding, Amin Saberi, and Yinyu Ye. Price of correlations in stochastic optimization. *Operations Research*, 60(1):150–162, 2012. Preliminary version in SODA 2010.

3 Saeed Alaei. Bayesian combinatorial auctions: Expanding single buyer mechanisms to many buyers. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 512–521, 2011.

4 Saeed Alaei. Bayesian combinatorial auctions: Expanding single buyer mechanisms to many buyers. *SIAM Journal on Computing*, 43(2):930–972, 2014.

5 Yossi Azar, Ashish Chiplunkar, and Haim Kaplan. Prophet secretary: Surpassing the 1-1/e barrier. In *Proceedings of the 2018 ACM Conference on Economics and Computation, Ithaca, NY, USA, June 18-22, 2018*, pages 303–318, 2018.

6 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011.

7 Shuchi Chawla, Jason D. Hartline, David L. Malec, and Balasubramanian Sivan. Multi-parameter mechanism design and sequential posted pricing. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 311–320, 2010.

8 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *SIAM J. Comput.*, 43(6):1831–1879, 2014.

9 Soheil Ehsani, Mohammad Hajiaghayi, Thomas Kesselheim, and Sahil Singla. Prophet secretary for combinatorial auctions and matroids. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms.* SIAM, 2018.

10 Hossein Esfandiari, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Morteza Monemizadeh. Prophet secretary. *SIAM Journal on Discrete Mathematics*, 31(3):1685–1701, 2017.

11 Moran Feldman, Ola Svensson, and Rico Zenklusen. Online contention resolution schemes. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1014–1033, 2016.

12 Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

13 Sudipto Guha and Kamesh Munagala. Approximation algorithms for budgeted learning problems. In *STOC*, pages 104–113. ACM, 2007. Full version as: *Approximation Algorithms for Bayesian Multi-Armed Bandit Problems*, `http://arxiv.org/abs/1306.3525`.

14 Anupam Gupta, Haotian Jiang, Ziv Scully, and Sahil Singla. The markovian price of information, 2018.

15 Anupam Gupta and Viswanath Nagarajan. A stochastic probing problem with applications. In *Integer Programming and Combinatorial Optimization - 16th International Conference, IPCO 2013, Valparaíso, Chile, March 18-20, 2013. Proceedings*, pages 205–216, 2013.

16 Anupam Gupta, Viswanath Nagarajan, and Sahil Singla. Algorithms and adaptivity gaps for stochastic probing. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1731–1747. SIAM, 2016.

17 Guru Guruganesh and Euiwoong Lee. Understanding the Correlation Gap For Matchings. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017)*, volume 93 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:15, 2018.

18 Mohammad Taghi Hajiaghayi, Robert D. Kleinberg, and Tuomas Sandholm. Automated online mechanism design and prophet inequalities. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 58–65, 2007.

19 Theodore P Hill and Robert P Kertz. A survey of prophet inequalities in optimal stopping theory. *Contemp. Math*, 125:191–207, 1992.

20 Robert Kleinberg and S. Matthew Weinberg. Matroid prophet inequalities. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 123–136, 2012.

21 Ulrich Krengel and Louis Sucheston. Semiamarts and finite values. *Bull. Am. Math. Soc*, 1977.

22 Ulrich Krengel and Louis Sucheston. On semiamarts, amarts, and processes with finite value. *Advances in Prob*, 4:197–266, 1978.

23 Brendan Lucier and Allan Borodin. Price of anarchy for greedy auctions. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 537–553. Society for Industrial and Applied Mathematics, 2010.

24 Aviad Rubinstein. Beyond matroids: secretary problem and prophet inequality with general constraints. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 324–332, 2016.

25 Aviad Rubinstein and Sahil Singla. Combinatorial prophet inequalities. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1671–1687. SIAM, 2017.

26 Qiqi Yan. Mechanism design via correlation gap. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 710–719. Society for Industrial and Applied Mathematics, 2011.

Equilibrium Computation in Atomic Splittable Routing Games

Umang Bhaskar[1]
Tata Institute of Fundamental Research, Mumbai, India
umang@tifr.res.in

Phani Raj Lolakapuri
Tata Institute of Fundamental Research, Mumbai, India
phaniraj@tcs.tifr.res.in
https://orcid.org/0000-0003-1593-9654

Abstract

We present polynomial-time algorithms as well as hardness results for equilibrium computation in atomic splittable routing games, for the case of general convex cost functions. These games model traffic in freight transportation, market oligopolies, data networks, and various other applications. An atomic splittable routing game is played on a network where the edges have traffic-dependent cost functions, and player strategies correspond to flows in the network. A player can thus split its traffic arbitrarily among different paths. While many properties of equilibria in these games have been studied, efficient algorithms for equilibrium computation are known for only two cases: if cost functions are affine, or if players are symmetric. Neither of these conditions is met in most practical applications. We present two algorithms for routing games with general convex cost functions on parallel links. The first algorithm is exponential in the number of players, while the second is exponential in the number of edges; thus if either of these is small, we get a polynomial-time algorithm. These are the first algorithms for these games with convex cost functions. Lastly, we show that in general networks, given input C, it is NP-hard to decide if there exists an equilibrium where every player has cost at most C.

2012 ACM Subject Classification Theory of computation → Network games

Keywords and phrases Routing Games, Equilibrium Computation, Convex costs, Splittable flows

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.58

Related Version A full version of the paper is available at [6], `https://arxiv.org/abs/1804.10044`.

1 Introduction

The problem of equilibrium computation, particularly efficient computation, is the cornerstone of algorithmic game theory, and is an area where researchers have had many successes. In many games, we have a good understanding of where the boundaries of computation lie, including normal-form games [9], markets [11], and congestion games [10]. The study of equilibrium computation has had a significant impact on algorithms, contributing new techniques and complexity classes.

[1] This work was partly funded by a Ramanujan Fellowship.

© Umang Bhaskar and Phani Raj Lolakapuri;
licensed under Creative Commons License CC-BY

In this paper, we are interested in equilibrium computation in atomic splittable routing games (ASRGs) with convex cost functions. These games are used to model many applications, including freight transportation, market oligopolies, and data networks (e.g., [8, 21]). In an ASRG, we are given a network with cost functions on the edges, and k players. Each player i has a source s_i, destination t_i, and a fixed demand v_i. Each player needs to transport its demand from its source to its destination at minimum cost, and is free to split the demand along multiple paths. Each player thus computes a minimum cost s_i-t_i flow, given the strategy of the other players.

The fact that each player can split its flow along multiple paths is what differentiates these from weighted congestion games. This freedom reduces the combinatorial structure of the game, making ASRGs harder to analyze. For example, equilibria in ASRGs may be irrational. While equilibrium computation in (unsplittable) congestion games is well-studied, much less is known about ASRGs. In fact, properties of ASRGs apart from equilibrium computation have been studied. We know tight bounds on the price of anarchy [8, 14, 25], and can characterize games with multiple equilibria [4].

However for equilibrium computation little is known. We know of only two cases when an equilibrium can be efficiently computed – when cost functions are affine, or when players are symmetric, i.e., they have the same source, destination, and demand [8, 16]. These conditions are hardly ever met in practice. We know of no hardness results for this problem. A number of iterative algorithms for equilibrium computation are proposed, and sufficient conditions for convergence are given by Marcotte [20]. Further, it is implicit in a paper by Swamy that one can compute equilibrium efficiently, given the total flow on each edge [27].

Computing equilibria in ASRGs is an interesting theoretical challenge as well. In some regards, properties of pure Nash equilibria in ASRGs resemble mixed Nash equilibria in games. For example, an equilibrium in pure strategies always exists [23]. For many games with this property, local search algorithms are known that converge to an equilibrium (e.g., congestion games). However in ASRGs we do not know of any such algorithms.

In this work, we focus on polynomial time algorithms for computing equilibria in ASRGs with general convex costs on parallel edges. Parallel edges are interesting because a number of applications can be modeled using parallel edges, such as load balancing across servers [26], and in traffic models [15]. Further, many results were first obtained for graphs consisting of parallel edges and then extended (e.g., results on the price of collusion [17], extended to series-parallel graphs [5], or on the price of anarchy [19, 7]). These are thus a natural starting point to study equilibrium computation. We believe it likely that some of our structural results extend beyond parallel edges, to nearly-parallel and series-parallel graphs.

Our Contribution. For ASRGs with convex costs on parallel edges, we give two algorithms. Our first algorithm computes an equilibrium[2] in time $O\left((\log|\mathcal{I}|)^n\right)$, where $|\mathcal{I}|$ is the input size and n is the number of players. If the number of players is near-logarithmic in the input size, i.e., $O(\log|\mathcal{I}|/\log\log|\mathcal{I}|)$, this gives a polynomial time algorithm. Our algorithm is based on the idea of reducing equilibrium computation to guessing the marginal costs of the players at equilibrium. The marginal costs turn out to have a number of interesting monotonicity properties, which we use to give a high-dimensional binary search algorithm.

[2] We use the standard notion of polynomial-time computation when outputs are possibly irrational: we say an algorithm is efficient if for any $\epsilon > 0$, the algorithm computes and ϵ-approximate solution in time polynomial in the inputs size and $\log(1/\epsilon)$.

Our second algorithm has running time exponential in the number of edges in the network. If the number of edges is constant, then this gives us a polynomial-time algorithm. Define players to be of the same *type* if they have flow on the same set of edges at equilibrium. The algorithm is based on the following structural result: for parallel edges, computing equilibrium in a general ASRG can be reduced to computing equilibrium in an ASRG where players of the same type also have the same demand. At a high level, this allows us to replace players of the same type by a single player, and then use our previous algorithm. Somewhat surprisingly, this result does not subsume the previous result. This is because the actual partition of players into types is unknown, hence we must enumerate over all possible such partitions, which introduces a factor of $O(n^{|E|})$ to the running time.

Lastly, we show that in general networks, determining existence of a Nash equilibrium where the cost of every player is at most C is NP-hard. Our proof here is a reduction from SUBSET-SUM, and builds upon a construction showing multiplicity of equilibria in ASRGs [4]. Our result parallels early results for bimatrix games [12], which showed that it is NP-hard to determine existence of a Nash equilibrium in bimatrix games where the cost of players is above a threshold [12]. Our proof is computer-assisted, and we use Mathematica to verify properties of equilibria in the games used in our reduction.

The complete proofs can be found in the full version of the paper [6].

Related Work. Existence of equilibria in atomic splitable routing games (and for more general *concave games*) is shown by Rosen [23]. Equilibria in these games are unique when delay functions of the edges are polynomials of degree ≤ 3 [2], when the players are symmetric, or when the underlying network is nearly-parallel [4]. In general, the equilibria may not be unique [4]. For computation, the equilibria can be obtained as the solution to a convex problem if the edge costs are linear, or if the players are symmetric [8]. Huang considers ASRGs with linear delays on a class of networks called *well-designed* which includes series-parallel graphs, and gives a combinatorial algorithm to find the equilibrium [18]. A network is well-designed if for the optimal flow (which minimizes total cost), increasing the total flow value does not decrease the flow on any edge. Recently, Harks and Timmermans gave an algorithm to compute equilibrium for ASRGs with player specific-linear costs on parallel links [16]. This setting allows players to have different cost functions on an edge. Their results use a reduction to integrally splittable flows, where the flow each player puts on an edge is an integral multiple of some quantity. In our case, the equilibrium flow can be irrational, hence these ideas do not seem to work. A number of algorithms for equilibrium computation are also proposed by Marcotte, based on either sequential best-response by the players, or linearization of the cost functions [20]. Marcotte shows conditions under which these algorithms converge to an equilibrium. It is unclear if these algorithms can be shown to run in polynomial time.

Nonatomic games have an infinite set of players, each of which has infinitesimal flow. Unlike ASRGs, equilibria in nonatomic games are well-studied: an equilibrium can be obtained by solving a convex program, and is unique if the costs are strictly increasing [3]. It is also known that ASRGs captures the setting where nonatomic players to form coalitions, and within a coalition players cooperate to minimize the total cost of its flow [17]. A number of papers study the change in total cost as players in a nonatomic game form coalitions, forming an ASRG [17, 5, 18]. Another property of ASRGs that has received a lot of attention is the price of anarchy (PoA), formalised as the ratio of the total cost of the worst equilibrium, to the optimal cost. Upper bounds on the PoA were obtained by Cominetti, Correa, and Stier-Moses [8] and improved upon by Harks [14]. These bounds were shown to be tight [25].

Atomic games where demands are unsplittable are also extensively studied. If all players have the same demand, these are called congestion games. Player strategies may not correspond to paths in a graph; if they do, these are called network congestion games. For congestion games, existence of a potential function is well-known [24], though computing an equilibrium is PLS-hard [10, 1], even if players are symmetric or the edge cost functions are linear. For symmetric players in a network congestion game, or if player strategies correspond to bases of a matroid, the equilibrium can be computed in polynomial time [1, 10].

2 Preliminaries

An *atomic splittable routing game* (ASRG) $\Gamma = (G = (V, E), (v_i, s_i, t_i)_{i \in [n]}, (l_e)_{e \in E}))$ is defined on a directed network $G = (V, E)$ with n players. Each player i wants to send v_i units of flow from s_i to t_i, where v_i is the *demand* of player i. Each edge e has a cost function $l_e(x)$ which is non-negative, increasing, convex and differentiable. Players are indexed so that $v_1 \geq v_2 \geq \ldots \geq v_n$, and the total demand $V := \sum_i v_i$. Vector f^i denotes the flow of player i. By abuse of notation, we say vector f is the flow on the network with n players such that f_e^i denotes the amount of flow player i sends along the edge e, and $f_e := \sum_i f_e^i$ is the total flow on edge e.

Given a flow $f = (f^1, f^2, \ldots, f^n)$ for n players, player i incurs a cost $\mathcal{C}_e^i(f) := f_e^i l_e(f_e)$ on the edge e. His total cost is $\mathcal{C}^i(f) = \sum_{e \in E} \mathcal{C}_e^i(f)$. Each player's objective is to minimize his cost, given the flow of the other players. We say a flow f is at *equilibrium*[3] if no player can unilaterally change his flow and reduce his total cost. More formally,

▶ **Definition 1.** In an ASRG a flow $f = (f^1, f^2, \ldots, f^n)$ is a Nash Equilibrium flow if for every player i and every flow $g = (f^1, f^2, \ldots, f^{i-1}, g^i, f^{i+1}, \ldots, f^n)$, where g^i is a flow of value v_i, $\mathcal{C}^i(f) \leq \mathcal{C}^i(g)$.

The equilibrium flow can be characterized in terms of the marginal costs of each player. Intuitively, the marginal cost for a player on a path is the increase in cost for the player when he increases his flow on the path by a small amount.

▶ **Definition 2.** Given a flow f, the marginal cost for the player i on path p is given by

$$L_p^i(f) = \sum_{e \in p} l_e(f_e) + f_e^i l_e'(f_e)$$

By applying the *Karush-Kuhn-Tucker* conditions [13] for player i's minimization problem, we get the following lemma 3 which characterizes the equilibrium using marginal costs.

▶ **Lemma 3.** *Flow $f = (f^1, f^2, \ldots, f^n)$ is a Nash equilibrium flow iff for any player i and any two directed paths p and q between s_i and t_i such that $f_e^i > 0$ $\forall e \in p$, $L_p^i(f) \leq L_q^i(f)$.*

Lemma 3 says that for any player i at equilibrium, the marginal delay $L_p^i(f)$ on all paths p such that $f_e^i > 0$ $\forall e \in p$ is equal, and is the minimum over all s_i-t_i paths. In a network of parallel edges, every edge is an s-t path, hence the condition holds at equilibrium with edges replacing paths.

We will frequently use the *support* of a player, where given a flow $f = (f^1, f^2, \ldots, f^n)$, the support of player i, S_i is defined as the set edges with $f_e^i > 0$.

[3] More specifically, a pure Nash equilibrium.

Swamy studies the use of edge tolls to enforce a particular flow as equilibrium [27]. However, if we start with an equilibrium flow, the tolls required are identically zero. The following theorem regarding equilibrium computation is then implicit, and will be useful to us in Section 4.

▶ **Theorem 4** ([27]). *For ASRG Γ, let $(h_e)_{e\in E}$ be the total flow on each edge at an equilibrium. Then given the total flow h, the equilibrium flow for each player can be obtained in polynomial time by solving a convex quadratic program.*

We make the following smoothness assumptions on edge cost functions.

1. Cost functions are continuously differentiable, nonnegative, convex, and increasing.
2. There is a constant $\Psi \geq V$ that satisfies:

$$\Psi \geq \max_{e\in E,\, x\in[0,V]} \left\{ l_e(x),\, l'_e(x),\, l''_e(x),\, \frac{1}{l'_e(x)} \right\}$$

By the first assumption, the edge marginal cost $L_e^i(f)$ is strictly increasing, both with the total flow f_e and player i's flow f_e^i. Define $L^i(f) := \min_{e\in E} L_e^i(f)$. Hence if $\vec{f}$, $\vec{f}'$ are two equilibrium flows, and for some player i and edge e $f_e \geq f'_e$, $f_e^i \geq {f'}_e^i$, and $f_e^i > 0$, then

$$L^i(f) = L_e^i(f) \geq L_e^i(f') \geq L^i(f')$$

and the second inequality is strict if $f_e > f'_e$ or $f_e^i > {f_e^i}'$. We frequently use this inequality in our proofs.

Also observe that for each edge e and flow values $x, y \in [0, V]$, the following properties of the edge cost functions hold.

$$|x-y| \ \leq\ \delta \Rightarrow\ |l_e(x) - l_e(y)| \leq \delta\Psi \quad \text{and} \quad |l_e(x) - l_e(y)| \ \leq\ \delta\ \Rightarrow\ |x-y| \ \leq\ \delta\Psi$$

3 An Algorithm with Complexity Exponential in the Number of Players

To convey the main ideas of the algorithm, we will ignore issues regarding finite precision computation in this section. In particular, we assume the algorithm carries out binary search to infinite precision, and show that such an algorithm computes the exact equilibrium. In this article we will give the algorithms for computing equilibria assuming that we can compute values upto arbitrary precision. In the full version of the paper [6], we give an implementation of the algorithm with finite precision. We show that our implementation computes an ϵ-equilibrium in time $O\left(mn^2\left(\log(n\Psi/\epsilon)\right)^n\right)$

Recall that the equilibrium in case of parallel edges is unique [22]. We start with an outline of the algorithm. Our first idea is to reduce the problem of equilibrium computation, to finding the *marginal costs* at equilibrium. We give a function GraphFlow that at a high level, given a vector of marginal costs $\vec{M} = (M^1, \ldots M^n)$, returns a vector of demands $\vec{w} = (w_1, \ldots, w_n)$ and a flow vector $\vec{f} = (f_e^i)_{e\in E, i\in[n]}$ so that (1) $\vec{f}$ is the equilibrium flow for the demand vector $\vec{w}$, and (2) for each player i, the marginal cost $L^i(f) = M^i$. That is, the marginal costs for the players at equilibrium are given by the input vector $\vec{M}$. We show that in fact each marginal cost vector $\vec{M}$ maps to a unique (demand, flow) pair that satisfies these conditions. Hence given marginal costs at equilibrium, the function must return the correct demands $(v_i)_{i\in[n]}$, and the required equilibrium flow. Thus, our problem reduces to finding a marginal cost vector $\vec{M}$ for which GraphFlow returns the *correct* demand vector $\vec{v} = (v^1, v^2, \ldots, v^n)$. We say a demand w^i for player i is *correct* if $w^i = v^i$.

Algorithm 1 GraphFlow($\vec{M}$).

```
Input: Vector $\vec{M} = (M^i)_{i\in[n]}$ of nonnegative real values
Output: Flow $\vec{f} = (f_i(e))_{i\in[n],e\in E}$ and demands $\vec{w} = (w_i)_{i\in[n]}$ so that $w_i = |f^i|$ and $\vec{f}$ is an
    equilibrium flow for demands $\vec{w}$ with marginal costs $\vec{M}$.
 1: Assume that $M^1 \geq M^2 \geq \cdots \geq M^n$, else renumber the vector components so that this
    holds.
 2: for each edge $e \in E$ do
 3:     $f_e^i = 0$ for each player $i \in [n]$
 4:     if $l_e(0) \geq M^1$ then
 5:         $S_e \leftarrow \emptyset$; continue with the next edge
 6:     for $k = 1 \to n$ do
 7:         $S = [k]$
 8:         Let $x_e$ be the unique solution to $kl_e(x) + xl'_e(x) = \sum_{i\in S} M^i$    ▷ Since $l_e(x)$ is
    strictly increasing and convex, the solution is unique
 9:         $f_e^i = \frac{M^i - l_e(x_e)}{l'_e(x_e)}$ for each player $i \in S$    ▷ Note that $\sum_{i\in S} f_e^i = x_e$
10:         if ($f_e^i \geq 0$ for all $i \in S$) and ($k = n$ or $M^{k+1} \leq l_e(x_e)$) then
11:             $f_e \leftarrow x_e$, $S_e \leftarrow S$, continue with the next edge
12: $w_i \leftarrow \sum_e f_e^i$ for each player $i$; return $(\vec{f}, \vec{w})$
```

Since only the marginal costs and demands matter to us, we can think of GraphFlow as a function from marginal cost vectors to demand vectors. We then give a high-dimensional binary search algorithm that computes the required marginal cost vector. This proceeds in a number of steps. We first show that the function GraphFlow is continuous, and is monotone in a strict sense: if we increase the marginal cost of a player, then the demand for this player increases, and the demand for every other player decreases. This allows us to show in Lemma 6 that given any marginal costs for the first $n-k$ players, there exist marginal costs for the remaining k players so that the demands returned by GraphFlow for these remaining players is correct. Lemma 6 allows us to ignore first $n-k$ players, and focus on the last k players, since no matter what marginal costs we choose for the first $n-k$ players, we can find marginal costs for the last k players that give the correct demand for these players.

The crux of our binary search algorithm is then Lemma 7, which says the following. Suppose we are given two marginal cost vectors $\vec{M}$ and $\vec{M'}$ that differ only in their last k coordinates, and for which the demands of the last $k-1$ players is equal. Thus, $M^i = M^{i'}$ for all players $i < k$, and the demands returned by GraphFlow($\vec{M}$), GraphFlow($\vec{M'}$) are equal for all players $i > k$. Suppose for the kth player, the demand with marginal costs $\vec{M}$ is higher than the demand with marginal costs $\vec{M'}$. Then Lemma 7 says that k's marginal cost in $\vec{M}$ must be higher than in $\vec{M'}$, i.e., $M^k > M^{k'}$. Thus Lemma 7 allows us to give a recursive binary search procedure. For a player k, the procedure fixes a marginal cost M^k, and finds marginal costs for players $i > k$ so that these players have the correct demand. By Lemma 6, we know that such marginal costs exist. With these marginal costs, if the demand for player k is greater than v_k, then by Lemma 7 M^k is too large. We then reduce M^k, and continue.

Algorithm 1 describes the function GraphFlow. The algorithm considers each edge in turn. For an edge e, it tries to find a subset of players $S \subseteq [n]$ and flows f_e^i so that, for all players $i \in S$, $L_e^i(f) = M^i$, and for all players not in S, $f_e^i = 0$ and $M^i \leq L_e^i(f)$. The set S can be obtained in $O(n)$ time by adding players to S in decreasing order of marginal costs

M^i. Given a set S, summing the equalities $L^i_e(f) = M^i$, we get the following equation with variable f_e:

$$|S|\, l_e(f_e) + f_e l'_e(f_e) \;=\; \sum_{i \in S} M^i\,.$$

Noting that the left-hand side is strictly increasing in f_e, we can solve this equation for f_e using binary search. This gives us the total flow on the edge f_e. We can then obtain the flow for each player by solving, for each player $i \in S$, the following equation:

$$f^i_e \;=\; \frac{M^i - l_e(f_e)}{l'_e(f_e)}\,.$$

We set $f^i_e = 0$ for all players not in S. It can be checked that $\sum_{i \in S} f^i_e = f_e$. If $f^i_e \geq 0$ for all players, and $L^i_e(f) = l_e(f_e) \geq M^i$ for all players not in S, we move on to the next edge. Else, we add the next player with lower marginal cost M^i to the set S, and recompute f_e.

We first establish in Claims 1, 2, and 3 that the algorithm is correct, and gives a continuous map from marginal cost vectors to demand vectors.

▶ **Claim 1.** *Given $\vec{M}$, assume w.l.o.g. that $M^1 \geq M^2 \geq \cdots \geq M^n$. Then GraphFlow($\vec{M}$) returns flow $\vec{f}$ and demands $\vec{w}$ so that, on each edge e and for each player i,*
1. *if $f^i_e = 0$ then $L^i_e(f) \geq M^i$*
2. *if $f^i_e > 0$ then $L^i_e(f) = M^i$*

Thus, $\vec{f}$ is an equilibrium flow for values $\vec{w}$, and if $w_i > 0$ then $M^i = L^i(f)$.

▶ **Claim 2.** *For each vector $\vec{M}$ of marginal costs, there is a unique pair of vectors $(\vec{w}, \vec{f})$ so that:*
1. *$\vec{f}$ is the equilibrium flow for demands $\vec{w}$, and*
2. *for each player i, $L^i(f) \geq M^i$. If $w_i > 0$, then $L^i(f) = M^i$.*

▶ **Corollary 5.** *For a demand vector $\vec{w}$, let $\vec{f}$ be the equilibrium flow, and $M^i = L^i(f)$ be the marginal costs of the players at equilibrium. Then the function GraphFlow($M^1, \ldots, M^n$) returns $(\vec{w}, \vec{f})$ as the output.*

▶ **Claim 3.** *Given marginal costs $\vec{M}$, $\vec{M}'$ so that for player l, $|M^l - M^{l'}| \leq \epsilon$, and $M^j = M^{j'}$ for all players $j \neq l$, let $(\vec{f}, \vec{w})$ and $(\vec{f}', \vec{w}')$ be the flows and demands returned by GraphFlow. Then for each player i, $|f^i_e - f^{i'}_e| \leq \epsilon'$, where $\epsilon' = 2n\Psi\epsilon$. Hence, for each player i, $|w_i - w'_i| \leq m\epsilon'$.*

Claim 3 then shows that the function GraphFlow is continuous.

In the remainder of the discussion, given a vector of marginal costs $\vec{M}$, we will primarily be concerned with the demands $\vec{w}$ returned by the function GraphFlow. We therefore define the functions GraphVal$_i$ for each player $i \in [n]$. Function GraphVal$_i$ takes as input a vector $\vec{M}$ of marginal costs for the players, and returns the demand w_i, the ith component of the demand vector $\vec{w}$ returned by GraphFlow($\vec{M}$). Claim 4 now shows that the function GraphFlow is monotone: if we increase the input marginal cost of a player, that player's demand goes up, while the demand for all the other players goes down. This is crucial in establishing existence of marginal costs for a subset of players (Lemma 6), and in our binary search algorithm later on.

▶ **Claim 4.** *Consider marginal cost vectors $\vec{M}$ and $\vec{M}'$ that differ only in their k^{th} coordinate, so that $\vec{M} = (M^i)_{i \in [n]}$ and $\vec{M}' = (M^1, M^2, \ldots, M^{k'}, \ldots, M^n)$. For each player i, let $w_i = GraphVal_i(\vec{M})$, and $w'_i = GraphVal_i(\vec{M}')$. If $M^{k'} > M^k$, then the following hold true as well:*

1. *$w'_k \geq w_k$,*
2. *if $w'_k > 0$, then $w'_k > w_k$,*
3. *$w'_i \leq w_i$ for $i \neq k$, and*
4. *for any subset of players P containing player k, $\sum_{i \in P} w'_i \geq \sum_{i \in P} w_i$.*

Consider the game with cost functions as in Γ, but with n players, each with demand V. Since this is a symmetric game, the equilibrium flow can be computed in polynomial time [8]. We define Λ to be the marginal cost of each player at this equilibrium.

▶ **Lemma 6.** *Let $S \subseteq [n]$ be a subset of the players. Given strictly positive input demands $\hat{w}_i \leq V$ for players $i \in S$ and marginal costs $M^i \leq \Lambda$ for players $i \notin S$, there exist marginal costs $\hat{M}^i \leq \Lambda$ for the players in S so that, given input $((\hat{M}^i)_{i \in S}, (M^i)_{i \notin S})$, $GraphVal_i$ returns $\hat{w}_i$ as the demand for players $i \in S$.*

▶ **Lemma 7.** *Given a player k, and two marginal cost vectors $\vec{M}$ and $\vec{M}'$ that satisfy the following properties:*
1. *for all players $i < k$, $M^i = M^{i'}$,*
2. *for all players $i > k$, $GraphVal_i(\vec{M}) = GraphVal_i(\vec{M}')$.*

Let $w_k = GraphVal_k(\vec{M})$, and $w'_k = GraphVal_k(\vec{M}')$. If $w_k > w'_k$, then $M^k > M^{k'}$.

Proof. Let $M^k \leq M^{k'}$, and let P be the set of players $\{i \geq k : M^i \leq M^{i'}\}$. Thus $k \in P$. We will show that $w_k \leq w'_k$. The proof proceeds by changing the marginal cost of each player in order from M^i to $M^{i'}$, and considering the effect on total demand of players in P. We show that in this process, the total demand of these players does not increase, and hence

$$\sum_{i \in P} \mathrm{GraphVal}_i(\vec{M}) \;\leq\; \sum_{i \in P} \mathrm{GraphVal}_i(\vec{M}')\,. \tag{1}$$

The expression on the left equals $w_k + \sum_{i \in P, i \neq k} w_i$, while the expression on the right equals $w'_k + \sum_{i \in P, i \neq k} w_i$ since for players $i > k$, $\mathrm{GraphVal}_i(\vec{M}) = \mathrm{GraphVal}_i(\vec{M}')$. Hence, this will show that $w_k \leq w'_k$, as required.
We now need to prove (1). Our proof uses Claim 4. Formally, let $\vec{M}(t) = ((M^{i'})_{i \leq t}, (M^i)_{i > t})$. Then $\vec{M}(0) = \vec{M}$, and $\vec{M}(n) = \vec{M}'$. We will show that

$$\sum_{i \in P} \mathrm{GraphVal}_i(\vec{M}(t)) \;\leq\; \sum_{i \in P} \mathrm{GraphVal}_i(\vec{M}(t+1))\,. \tag{2}$$

For each player $t \in [n]$, there are three cases: either (1) $M^t = M^{t'}$, (2) $M^t \leq M^{t'}$ and $t \in P$, or (3) $M^t > M^{t'}$ and $t \notin P$. We consider these three cases separately. In the first case, $\vec{M}(t) = \vec{M}(t+1)$, and (2) clearly holds. In the second case, by Claim 4 and since $t \in P$, (2) holds. In the third case, again by Claim 4, for all $i \neq t$ the demand either increases or remains the same. Since $t \notin P$, the total demand of players in P either increases or remains the same, and hence (2) holds. This completes the proof. ◀

We now give our algorithm for obtaining the "correct" marginal costs $\vec{M}$, so that $\mathrm{GraphFlow}(\vec{M})$ returns the required equilibrium flow. The algorithm is recursive. For each player k, it picks a candidate marginal cost M^k, and then recursively calls itself to find marginal costs M^i for players $i > k$ so that the demands for these players $i > k$ is correct. If the demand for player k itself is too large, it reduces M^k, and otherwise increases M^k. Thus the algorithm conducts a binary search to find the correct marginal cost for player k, and in each iteration calls itself to determine correct marginal values for players $i > k$.

Algorithm 2 $\text{EqMCost}_k((M^1, \dots, M^{k-1}))$.

```
Input: Vector (M^1,...,M^{k-1}), with each component M^i ∈ [0,Λ]    ▷ If k = 1, there is no input required.
Output: Vector ($\vec{M}$) of marginal costs so that the first $k-1$ marginal costs are equal to
    the inputs, and for players $i \geq k$, the demand $\text{GraphVal}_i(\vec{M}) = v_i$.
1:  if $k = n$ then
2:      Using binary search in $[0, \Lambda]$, find $M$ so that $\text{GraphVal}_n((M^i)_{i<n}, M) = v_n$. return $M$.
3:  $\text{Low} \leftarrow 0$, $\text{High} \leftarrow \Lambda$, $\text{Mid} \leftarrow (\text{Low} + \text{High})/2$
4:  $(M^{k+1}, \dots, M^n) \leftarrow \text{EqMCost}_{k+1}(M^1, \dots, M^{k-1}, \text{Mid})$
    ▷ Call $\text{EqMCost}_{k+1}$ to get marginal costs for the remaining player $k+1, \dots, n$ so that
    the demand for these players is correct
5:  if $(\text{GraphVal}_k((M^i)_{i<k}, \text{Mid}, (M^i)_{i>k}) = v_k)$ then
6:      return $(\text{Mid}, (M^i)_{i>k})$
7:  else if $(\text{GraphVal}_k((M^i)_{i<k}, \text{Mid}, (M^i)_{i>k}) > v_k)$ then
8:      $\text{High} \leftarrow \text{Mid}$, $\text{Mid} \leftarrow (\text{Low} + \text{High})/2$, goto 4
9:  else if $(\text{GraphVal}_k((M^i)_{i<k}, \text{Mid}, (M^i)_{i>k}) < v_k)$ then
10:     $\text{Low} \leftarrow \text{Mid}$, $\text{Mid} \leftarrow (\text{Low} + \text{High})/2$, goto 4
```

▶ **Theorem 8.** *For ASRG Γ, $EqMCost_1$ returns marginal cost vector $\vec{M}$ such that $GraphFlow(\vec{M}) = (\vec{w}, \vec{f})$, where $\vec{w} = \vec{v}$ and $\vec{f}$ is the equilibrium flow in Γ.*

The main ingredient in the proof of Theorem 8 is the following lemma, which shows that recursively, for any player k, the function EqMCost returns correct marginal costs.

▶ **Lemma 9.** *For any vector $(M^1, \dots, M^{k-1})$ with each component in $[0, \Lambda]$, the function $EqMCost_k((M^1, \dots, M^{k-1}))$ returns marginal costs $(M^k, \dots, M^n)$ for the remaining players so that, for each player $i \geq k$, $GraphVal_i(M^1, \dots, M^n) = v_i$.*

Proof. The proof is by induction on n. In the base case, $k = n$, and the input is the vector $(M^1, \dots, M^{n-1})$ with each component in $[0, \Lambda]$. By Lemma 6, there exists $\hat{M}$ so that $\text{GraphVal}_n(M^1, \dots, M^{n-1}, \hat{M}) = v_n$. We now show that the value $\hat{M}$ can correctly be found by binary search. Initially, the search interval is $[0, \Lambda]$, and by Lemma 6, $\hat{M}$ lies in the search interval. Assume in some iteration the search interval is $[\text{Low}, \text{High}]$; $\hat{M}$ lies in the search interval; and that $\text{GraphVal}_n(M^1, \dots, M^{n-1}, \text{Mid}) > v_n$. Since $v_n > 0$, and $v_n = \text{GraphVal}_n(M^1, \dots, M^{n-1}, \hat{M}) < \text{GraphVal}_n(M^1, \dots, M^{n-1}, \text{Mid})$, it follows by Lemma 7 that $\hat{M} < \text{Mid}$. Hence, $\hat{M}$ lies in the interval $[\text{Low}, \text{Mid}]$, and we can restrict our search to this space, which is exactly how the binary search proceeds. The case when $\text{GraphVal}_n(M^1, \dots, M^{n-1}, \text{Mid}) < v_n$ is similar, and $\hat{M}$ then lies in the interval $[\text{Mid}, \text{High}]$.

For the inductive step, we are given player $k < n$. We assume that given any input vector $(M^1, \dots, M^k)$ with each component in $[0, \Lambda]$, EqMCost_{k+1} returns marginal costs $(M^{k+1}, \dots, M^n)$ for the remaining players so that for each of these remaining players $i \geq k+1$, $\text{GraphVal}_i(M^1, \dots, M^n) = v_i$. We need to show that given any input marginal costs $(M^1, \dots, M^{k-1})$ for the first $k-1$ players, EqMCost_k finds marginal costs $(M^k, \dots, M^n)$ for players k onwards so that the demand returned for these players $i \geq k$ by GraphVal_i is v_i. Firstly, by Lemma 6, choosing $S = [k, \dots, n]$ and $\hat{w}_i = v_i$ for players $i \in S$, there exist marginal costs $(\hat{M}^k, \dots, \hat{M}^n)$ so that for all $i \geq k$, $\text{GraphVal}_i((M^i)_{i<k}, (\hat{M}^i)_{i \geq k}) = v_i$. We now show that the binary search procedure in EqMCost_k finds the required marginal cost $\hat{M}^k$. By Lemma 6, $\hat{M}^k$ lies in the initial search interval $[0, \Lambda]$. Assume that in some iteration,

$\hat{M}^k$ lies in the search interval $[\text{Low}, \text{High}]$, and $\text{Mid} = (\text{Low} + \text{High})/2$. By the induction hypothesis, $\text{EqMCost}_{k+1}(M^1, \dots, M^{k-1}, \text{Mid})$ returns marginal costs $(M^{k+1}, \dots, M^n)$ for the players $k+1, \dots, n$ so that for each of these players $i \geq k+1$ (but not player k), $\text{GraphVal}_i((M^i)_{i<k}, \text{Mid}, (M^i)_{i>k}) = v_i$. Further, for each player $i \geq k$, $\text{GraphVal}_i((M^i)_{i<k}, (\hat{M}^i)_{i \geq k}) = v_i$. Suppose that for player k, $\text{GraphVal}_k((M^i)_{i<k}, \text{Mid}, (M^i)_{i>k}) > v_k = \text{GraphVal}_i((M^i)_{i<k}, (\hat{M}^i)_{i \geq k})$. Then by Lemma 7, $\text{Mid} > \hat{M}^k$, and hence $\hat{M}^k$ lies in the interval $[\text{Mid}, \text{High}]$. The algorithm then reduces the search space to this interval, and continues. If $\text{GraphVal}_k((M^i)_{i<k}, \text{Mid}, (M^i)_{i>k}) < v_k$, it can be similarly shown that $\text{Mid} < \hat{M}^k$. Thus, $\hat{M}^k$ always lies in the search space $[\text{Low}, \text{High}]$, which is halved in each iteration. The binary search is thus correct, and must eventually terminate. ◀

Proof of Theorem 8. By Lemma 9, EqMCost_1 returns a marginal cost vector $\vec{M} = (M^1, \dots, M^n)$ so that $\text{GraphVal}_i(\vec{M}) = v_i$ for each player i. Let $\vec{v} = (v_1, \dots, v_n)$. By definition of the function GraphVal, this implies that $\text{GraphFlow}(\vec{M})$ returns vectors $\vec{v}$ and $\vec{f}$. Finally by Claims 1 and 2, $\vec{f}$ is the equilibrium flow for demands $\vec{v}$, as required. ◀

Implementation and Complexity. Under the assumption that binary search could be done to arbitrary precision, we showed that the algorithm EqMCost is correct. However, the solutions to the polynomial equations could be irrationals, and thus the algorithm given is not a finite algorithm. In the full version of paper [6], we show that for any given error parameter ϵ, we can implement EqMCost to run in time $O\left(\text{poly}(\log \Psi, \log \frac{1}{\epsilon}, m, n)\right)$, and return an ϵ-equilibria. We say a flow $\vec{f}$ is an ϵ-equilibrium if any player i has flow only on ϵ-minimum marginal cost edges. That is, if $f_e^i > 0$ for player i on edge e, then $L_e^i(f) \leq \min_{e'} L_{e'}^i(f) + \epsilon$. Conventionally, a strategy profile is an ϵ-equilibrium if no player can improve it's cost by ϵ. One can check that the two are equivalent: if a flow f is an ϵ-equilibrium by our definition, then no player i can improve its cost by more than ϵv_i, where v_i is its demand.

The work in giving an implementation for the algorithm EqMCost is in implementing the binary search correctly, up to some error parameter ϵ. Since the algorithm is iterative, this error grows across each iteration, and bounding the error in each iteration is quite technical. Additionally, approximate versions of some of the results for EqMCost have to be reproved. Further details are given in the full version of paper [6].

4 An Algorithm with Complexity Exponential in Number of Edges

Our second algorithm is based on Theorem 10, which shows that at equilibrium the supports of players form chains.

▶ **Theorem 10** ([4]). *Consider an ASRG with n players on a graph consisting of parallel edges*[4]*, and let f be the equilibrium flow. Then $L^1(f) \geq \dots \geq L^n(f)$. Consequently, the supports $S_1(f) \supseteq \dots \supseteq S_n(f)$.*

Thus in an ASRG on m parallel links, there exist numbers $1 = a_1 < a_2 < \dots < a_T \leq n$ with $T \leq m$, so that players with indices in $[a_i, a_{i+1} - 1]$ have the same support at equilibrium. Define a *type set* $\mathcal{T} = (P_1, \dots, P_T)$ with $T \leq m$ to be a partition of the players so that players in a set P_t in the partition have consecutive indices. Hence, a type set $\mathcal{T} = (P_1, \dots, P_T)$ can be denoted by a sequence of numbers $(a_1, \dots, a_T)$ where $1 = a_1 < a_2 < \dots < a_T \leq n$ and P_t consists of the players with indices $a_t, \dots, a_{t+1} - 1$. We say a type set is *valid* for

[4] The proof by Bhaskar et al. [4] is for series-parallel graphs which are a superset of parallel link graphs.

Γ iff two players in the same partition in $\mathcal{T}$ also have the same support in the equilibrium. Theorem 10 then shows that in a graph consisting of m parallel links, there is a type set that is valid.

We will now give an algorithm with running time that is exponential in the number of edges, using the algorithm from Section 3, which is exponential in the number of players, and Theorem 4. Our algorithm in this section crucially uses Lemma 11, which has the following content. Let $\mathcal{T} = (P_1, \ldots, P_T)$ be a type set, not necessarily valid, for a game Γ. Consider the game $\Gamma^{\mathcal{T}}$ where for each set $P_t \in \mathcal{T}$, we replace the players in P_t with $|P_t|$ players that have the same demand, given by $\sum_{i \in P_t} v_i / |P_t|$. That is, we pick a set P_t, and replace all players in this set by players with demands equal to the average demand of players in P_t. We do this for each set P_t. Lemma 11 then says that if $\mathcal{T}$ is valid for Γ, then the total flow on any edge does not change between Γ and $\Gamma^{\mathcal{T}}$.

▶ **Lemma 11.** *Let $\mathcal{T} = (P_1, \ldots, P_T)$ be the valid type set for game Γ with n players on a network of parallel edges. Let f and g be the respective equilibrium flows for games Γ and $\Gamma^{\mathcal{T}}$ respectively. Then on each edge e, $f_e = g_e$.*

The proof of Lemma 11 closely follows an earlier proof of uniqueness of equilibrium in an ASRG [4, Theorem 5]. Lemma 11 implies that in an ASRG on parallel edges we can replace players that have the same support with players that have the same demand without affecting the total flow at equilibrium on the edges. Further, given the total flow on each edge at equilibrium in a general network, the flow for each player can be computed by Theorem 4.

The algorithm EqMCost runs a binary search for the marginal cost at equilibrium for each player. For each player i, EqMCost$_i$ runs a binary search, in each iteration of which it calls EqMCost$_{i+1}$. If each binary search runs for R iterations, and each iteration takes time S, then this recursively gives a running time of $O(S\,R^n)$. However, if players i, $i+1$ have the same demand, then by Theorem 10 they also have the same marginal cost at equilibrium. Hence we can run the binary search for the marginal cost at equilibrium, for both players *simultaneously*. This is the basic idea for the modification. For a given type set $\mathcal{T} = (P_1, \ldots, P_T)$, and the game $\Gamma^{\mathcal{T}}$, we know that all players in the same set P_t have the same marginal cost at equilibrium. Hence, we run the binary search once for each set P_t, rather than once for each player. By Theorem 10, the number of types $T \le m$.

This gives us the reduced running time of $O(S\,R^m)$ for type set $\mathcal{T}$, for each run of the modified algorithm EqMCostTy. However, note that we run the algorithm once for each possible type set. If the game is played on m parallel links, then the number of possible type sets is about $(n+m)^m$. In the full version of paper [6], we show that this takes a running time of $O\left((n+m)^m m^3 \left(\log(n\Psi/\epsilon)\right)^m\right)$.

5 Hardness of Computing Equilibria

Prior to the proof of PPAD-hardness of computing a (mixed) Nash equilibrium in bimatrix games, Gilboa and Zemel showed that it was NP-hard to determine if there existed an equilibrium in bimatrix games where every player had payoff above a given threshold C [12]. We show a similar result for ASRGs.

▶ **Theorem 12.** *Given an ASRG with convex, strictly increasing and continuously differentiable cost functions, it is* NP-hard *to determine if there exists an equilibrium at which cost of every player is at most C.*

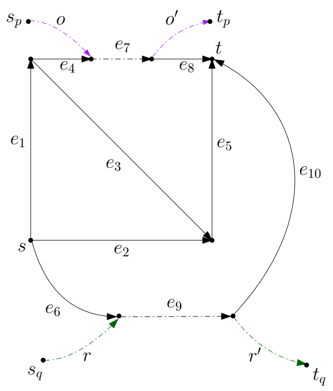


Figure 1 ASRG $\mathcal{G}$, with multiple equilibria.

The main idea of the proof is to build upon the existence of multiple equilibria in ASRGs, and is a reduction from SUBSET-SUM. In this problem, we are given a set $S = \{s_1, s_2, \ldots, s_n\} \subset \mathbb{N}$ such that the sum of elements in S is M, and we want to determine if there exists a subset $T \subseteq S$ such that the sum of elements in T is $M/2$. SUBSET-SUM problem is known to be NP-complete. Our reduction is in two steps. First, we construct an ASRG $\mathcal{G}$ with four players b, r, p, and q, and exactly three equilibria, one of which is irrational, which is used as gadget in the reduction (Figure 1).[5] This construction builds upon an earlier example showing multiplicity of equilibria in ASRGs [4]. We will be mainly concerned with the rational equilibrium flows, say f and g. We choose the cost functions so that (i) $\mathcal{C}^p_{e_7}(g) > \mathcal{C}^p_{e_7}(f)$, and (ii) the sum of costs of players p and q are equal for f and g, that is,

$$\Lambda \;:=\; \mathcal{C}^p_{e_7}(f) + \mathcal{C}^q_{e_9}(f) \;=\; \mathcal{C}^p_{e_7}(g) + \mathcal{C}^q_{e_9}(g) \tag{3}$$

Then $\mathcal{C}^q_{e_9}(f) > \mathcal{C}^q_{e_9}(g)$.

We now repeat this subgame n times in series, once for each element in the set S in the SUBSET-SUM instance. Each subgame is independent of the others, i.e., the players b_i and r_i in i^{th} subgame are local to that subgame and do not play any role in other subgames. All the subgames are connected by players p and q, who can only use one edge (e_7 and e_9 respectively) in each subgame. We show that f, g, and h continue to be the only equilibria within each subgame. In the ith subgame $\mathcal{G}_i$, we multiply each cost function by s_i. This causes all costs to get multiplied by s_i, and does not affect the equilibria. Thus, in each subgame $\mathcal{G}_i$, player p has costs $s_i\mathcal{C}^p_{e_7}(f)$ and $s_i\mathcal{C}^p_{e_9}(g)$ in equilibrium flows f, g, and h (confined to the subgame) respectively. Similar for player q. Roughly, we think of equilibrium f in a subgame as putting s_i in the subset S, and equilibrium g as leaving s_i out.

We will show that if the given instance satisfies SUBSET-SUM, then there exists an equilibrium at which both players p and q have cost $(M\Lambda)/2$, otherwise at least one of them has cost strictly greater than $(M\Lambda)/2$. At equilibrium in the game, let F be the subgames where f is the equilibrium, and G be the subgames where g is the equilibrium. Then the total cost of players p and q is

$$\mathcal{C}^p_{e_7}(f)\sum_{i\in F} s_i + \mathcal{C}^p_{e_7}(g)\sum_{i\in G} s_i + \mathcal{C}^q_{e_9}(f)\sum_{i\in F} s_i + \mathcal{C}^q_{e_9}(g)\sum_{i\in G} s_i \;=\; M\Lambda$$

[5] We use Mathematica to verify properties of equilibria in the games used in the reduction. Further details are given in the full version of paper [6].

where the equality follows from (3). From $\mathcal{C}^p_{e_7}(g) > \mathcal{C}^p_{e_7}(f)$, it follows that the cost of each player p, q is $(M\Lambda)/2$ iff at equilibrium, $\sum_{i\in F} s_i = \sum_{i\in G} s_i$. Else, since the sum of costs of the two players is constant, exactly one player has cost above $(M\Lambda)/2$.

To complete the proof, we add player-specific edges to ensure that $(M\Lambda)/2$ is large enough so that all the other players b_i, r_i always have cost at most $(M\Lambda)/2$ at any equilibrium.

References

1 Heiner Ackermann, Heiko Röglin, and Berthold Vöcking. On the impact of combinatorial structure on congestion games. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 613–622, 2006.

2 Eitan Altman, Tamer Başar, Tania Jimenez, and Nahum Shimkin. Competitive routing in networks with polynomial costs. *Automatic Control, IEEE Transactions on*, 47(1):92–96, 2002.

3 MJ Beckmann, CB Mc Guire, and CB Weinstein. Studies in the economics of transportation, Yale University Press. *New Haven, Connecticut, USA*, 1956.

4 Umang Bhaskar, Lisa Fleischer, Darrell Hoy, and Chien-Chung Huang. On the uniqueness of equilibrium in atomic splittable routing games. *Math. Oper. Res.*, 40(3):634–654, 2015. doi:10.1287/moor.2014.0688.

5 Umang Bhaskar, Lisa Fleischer, and Chien-Chung Huang. The price of collusion in series-parallel networks. In *Integer Programming and Combinatorial Optimization, 14th International Conference, IPCO 2010, Lausanne, Switzerland, June 9-11, 2010. Proceedings*, pages 313–326, 2010.

6 Umang Bhaskar and Phani Raj Lolakapuri. Equilibrium computation in atomic splittable routing games with convex cost functions. *CoRR*, abs/1804.10044, 2018. arXiv:1804.10044.

7 Kshipra Bhawalkar, Martin Gairing, and Tim Roughgarden. Weighted congestion games: The price of anarchy, universal worst-case examples, and tightness. *ACM Trans. Economics and Comput.*, 2(4):14:1–14:23, 2014.

8 Roberto Cominetti, José R. Correa, and Nicolás E. Stier Moses. The impact of oligopolistic competition in networks. *Operations Research*, 57(6):1421–1437, 2009. doi:10.1287/opre.1080.0653.

9 Constantinos Daskalakis. On the complexity of approximating a Nash equilibrium. *ACM Trans. Algorithms*, 9(3):23:1–23:35, 2013.

10 Alex Fabrikant, Christos H. Papadimitriou, and Kunal Talwar. The complexity of pure Nash equilibria. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 604–612, 2004.

11 Jugal Garg, Ruta Mehta, Vijay V. Vazirani, and Sadra Yazdanbod. Settling the complexity of Leontief and PLC exchange markets under exact and approximate equilibria. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 890–901, 2017.

12 Itzhak Gilboa and Eitan Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1(1):80–93, 1989.

13 Geoff Gordon and Ryan Tibshirani. Karush-Kuhn-Tucker conditions. *Optimization*, 10(725/36):725, 2012.

14 Tobias Harks. Stackelberg strategies and collusion in network games with splittable flow. *Theory of Computing Systems*, 48(4):781–802, 2011.

15 Tobias Harks, Ingo Kleinert, Max Klimm, and Rolf H Möhring. Computing network tolls with support constraints. *Networks*, 65(3):262–285, 2015.

16 Tobias Harks and Veerle Timmermans. Equilibrium computation in atomic splittable singleton congestion games. In *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, pages 442–454, 2017.

17 Ara Hayrapetyan, Éva Tardos, and Tom Wexler. The effect of collusion in congestion games. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 89–98, 2006.

18 Chien-Chung Huang. Collusion in atomic splittable routing games. *Theory Comput. Syst.*, 52(4):763–801, 2013. `doi:10.1007/s00224-012-9421-4`.

19 Elias Koutsoupias and Christos H. Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, 2009.

20 Patrice Marcotte. Algorithms for the network oligopoly problem. *Journal of the Operational Research Society*, pages 1051–1065, 1987.

21 Ariel Orda, Raphael Rom, and Nahum Shimkin. Competitive routing in multiuser communication networks. *IEEE/ACM Transactions on Networking (ToN)*, 1(5):510–521, 1993.

22 Oran Richman and Nahum Shimkin. Topological uniqueness of the Nash equilibrium for selfish routing with atomic users. *Math. Oper. Res.*, 32(1):215–232, 2007. `doi:10.1287/moor.1060.0229`.

23 J Ben Rosen. Existence and uniqueness of equilibrium points for concave n-person games. *Econometrica: Journal of the Econometric Society*, pages 520–534, 1965.

24 Robert W Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2(1):65–67, 1973.

25 Tim Roughgarden and Florian Schoppmann. Local smoothness and the price of anarchy in splittable congestion games. *Journal of Economic Theory*, 156:317–342, 2015.

26 Subhash Suri, Csaba D Tóth, and Yunhong Zhou. Selfish load balancing and atomic congestion games. *Algorithmica*, 47(1):79–96, 2007.

27 Chaitanya Swamy. The effectiveness of Stackelberg strategies and tolls for network congestion games. *ACM Trans. Algorithms*, 8(4):36:1–36:19, 2012. `doi:10.1145/2344422.2344426`.

Online Non-Preemptive Scheduling to Minimize Weighted Flow-time on Unrelated Machines

Giorgio Lucarelli
Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, France
giorgio.lucarelli@imag.fr

Benjamin Moseley
Carnegie Mellon University, USA
moseleyb@andrew.cmu.edu

Nguyen Kim Thang
IBISC, Univ Evry, University Paris-Saclay, France
thang@ibisc.fr

Abhinav Srivastav
IBISC, Univ Evry, University Paris-Saclay, France
abhinavsriva@gmail.com

Denis Trystram
Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, France
denis.trystram@imag.fr

Abstract

In this paper, we consider the online problem of scheduling independent jobs *non-preemptively* so as to minimize the weighted flow-time on a set of unrelated machines. There has been a considerable amount of work on this problem in the preemptive setting where several competitive algorithms are known in the classical competitive model. However, the problem in the non-preemptive setting admits a strong lower bound. Recently, Lucarelli et al. presented an algorithm that achieves a $O\left(\frac{1}{\epsilon^2}\right)$-competitive ratio when the algorithm is allowed to reject ϵ-fraction of total weight of jobs and has an ϵ-speed augmentation. They further showed that speed augmentation alone is insufficient to derive any competitive algorithm. An intriguing open question is whether there exists a scalable competitive algorithm that rejects a small fraction of total weights.

In this paper, we affirmatively answer this question. Specifically, we show that there exists a $O\left(\frac{1}{\epsilon^3}\right)$-competitive algorithm for minimizing weighted flow-time on a set of unrelated machine that rejects at most $O(\epsilon)$-fraction of total weight of jobs. The design and analysis of the algorithm is based on the primal-dual technique. Our result asserts that alternative models beyond speed augmentation should be explored when designing online schedulers in the non-preemptive setting in an effort to find provably good algorithms.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Online Algorithms, Scheduling, Resource Augmentation

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.59

Funding Nguyen Kim Thang is supported by the ANR project OATA n° ANR-15-CE40-0015-01, Hadamard PGMO and DIM RFSI. Abhinav Srivastav was supported by the PSL project Multi-Fac and is supported by the ANR project OATA n° ANR-15-CE40-0015-01, Hadamard PGMO and DIM RFSI. Benjamin Moseley was supported in part by a Google Research Award, and NSF Grants CCF-1617724, CCF-1733873 and CCF-1725661.

© Giorgio Lucarelli, Benjamin Moseley, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram; licensed under Creative Commons License CC-BY

1 Introduction

In this work, we study the fundamental problem of online scheduling of independent jobs on unrelated machines. Jobs arrive over time and the online algorithm has to make the decision which job to process *non-preemptively* at any time on each machine. A job j is released at time r_j and takes p_{ij} amount of processing time on a machine i. Further, each job has a weight w_j that denotes its (relative) priority. Our aim is to design a non-preemptive schedule that minimizes the total weighted flow-time (or response time) quantity, i.e., $\sum_j w_j(C_j - r_j)$ where C_j denotes the completion time of job j.

We are interested in designing online non-preemptive scheduling problem in the worst-case model. Several strong lower bounds are known for simple instances [2, 4]. The main hurdle arises from two facts: the algorithm must be online and robust to all problem instances and the algorithmic decisions made should be of irrevocable nature. In order to overcome the strong theoretical lower bound, Kalyanasundaram and Pruhs [7] and Phillips et al. [10] proposed the analysis of scheduling algorithms in terms of the speed augmentation and machine augmentation, respectively. Together these augmentation are commonly referred to as resource augmentation. Here, the idea is to either give the scheduling algorithm faster processors or extra machines in comparison to the adversary. For preemptive problems, these models provide a tool to establish a theoretical explanation for the good performance of algorithms in practice. In fact, many practical heuristics have been shown to be competitive where the algorithm is given resource augmentation. In contrast, problems in the non-preemptive setting have resisted against provably good algorithms even with such additional resources [9].

Choudhury et al. [5] proposed a new model of resource augmentation where the online algorithm is allowed to reject some of the arriving jobs, while the adversary must complete all jobs. Using a combination of speed augmentation and rejection, Lucarelli et al. [9] break this theoretical barrier and gave a scalable algorithm for non-preemptive weighted flow-time problems. However, it remains an intriguing question about the power of rejection model in comparison to the previous ones.

Recently, Lucarelli et al. [8] showed that a $O(1)$ competitive algorithm exists if all jobs have unit weight and one only rejects a constant fraction of the jobs. Their algorithm and analysis are closely tied to the unweighted case and there is no natural extension to the case where jobs have weights. The question looms, does there exist a constant competitive algorithm for non-preemptive scheduling to minimize weighted flow-time using rejection?

1.1 Our Result and Approach

This paper gives the first algorithm with non-trivial guarantees for minimizing weighted flow time using rejection and no other form of resource augmentation. The main result of the paper is the following theorem. The theorem shows that constant competitiveness can be achieved by only rejecting a small faction of the total weight of the jobs.

▶ **Theorem 1.** *For the non-preemptive problem of minimizing weighted flow-time on unrelated machines, there exists a $O(\frac{1}{\epsilon^3})$-competitive algorithm that rejects at most $O(\epsilon)$-fraction of total weight of the jobs for any $0 < \epsilon < 1$.*

The algorithmic decisions are classified into three parts: dispatching, rejecting and scheduling policy. The scheduling follows HDF policy (Highest Density First) once jobs are assigned to the machines. At the arrival of a job, for each machine, the algorithm computes an approximate increase in the weighted flow-time and assigns the job to the machine with the least increase in the approximate weighted flow-time. To compute this quantity for a given machine, the algorithm considers the set of uncompleted jobs in the machine queue in the non-increasing order of densities and uses two different rejection policies.

The first rejection policy, referred as the *preempt* rule, rejects jobs that have already started processing if the total weight of newly arrived "high priority" jobs (high density jobs) exceeds a given threshold. Specifically, when a job starts executing, we associate a counter that keeps tracks of the total weight of newly arrived jobs. Once the value of this counter is at least $1/\epsilon$ times the weight of the current executing job, the algorithm preempts the current executing job and rejects it. The rejected job is pushed out of the system so as to be never executed again.

We emphasize here a critical issue due to job rejection which is of different nature to speed augmentation. Observe that rejecting a job that has already started processing may cause a large decrease in the weighted flow-time of the jobs in the machine queue. Due to job arrivals and job rejections, quantities associated to the machine queue (for example the remaining job weight, etc) vary arbitrarily without any nice properties like monotonicity. That creates a significant challenge in the dual fitting analysis. To tackle this problem, we introduce the notion of *definitive completion time* for each job. Once a job is rejected or completed before its definitive completion time, the algorithm removes the job from the queue of the machine. However, for the purpose of analysis, the rejected jobs are still considered in the definition of dual variables until their definitive completion time. This ensures that for any fixed time, the weight of jobs not yet definitively completed increases with the arrival of new jobs (see Section 3.4 for details).

The second rejection policy, referred as the *weight-gap* rule, rejects unprocessed "low priority" jobs (small density jobs) from the machine's queue. This policy simulates the ϵ-speed-augmentation. In the particular case where all jobs have the same weight, this rejection policy rejects a "low priority" job for every $1/\epsilon$ arrivals of new jobs. Due to the scheduling policy, if a "low priority" job is not rejected, then it will be completed last in the schedule (assuming no future job arrivals).

In the algorithm's schedule, future arriving jobs do not delay the rejected low priority jobs, while the later ones need to be completed in the adversary's schedule. This is where the algorithm benefits from the power of rejection. Specifically, the algorithm can use the difference between the rejection time and the definitive completion time of jobs to create a similar effect to speed augmentation. The key idea is to reject the low priority jobs so their total weight is comparable to jobs that arrive after them.

The definitive completion times play a crucial role so that the dual achieves a substantial value compared to the primal. By carefully choosing the definitive completion times of jobs, we manage to prove the competitive ratio of our algorithm with admittedly sophisticated analysis.

1.2 Related Works

The problem of minimizing the total weighted flow-time has been extensively studied in the online scenario. For the preemptive problem, Chekuri et al. [4] presented a *semi-online* $O(\log^2 P)$-competitive algorithm for a single machine, where P is the ratio of the largest to the smallest processing time of the instance. Later, Bansal and Dhamdhere [3] proposed a $O(\log W)$-competitive algorithm, where W is the ratio between the maximum and the minimum weights of the jobs. This was later improved in [2]. In contrast to the single-machine case, Chekuri et al. [4] showed a $\Omega(\min(\sqrt{P}, \sqrt{W}, \frac{n}{m}^{\frac{1}{4}}))$ lower bound for m identical machines. For the online non-preemptive problem of minimizing the total weighted flow-time, Chekuri et al. [4] showed that any algorithm has at least $\Omega(n)$ competitive ratio for single machine where n is the number of jobs.

In speed-augmentation model, Anand et al. [1] presented a scalable competitive algorithm for the preemptive problem on a set of unrelated machines. For the non-preemptive setting, Phillips et al. [10] gave a constant competitive algorithm in identical machine setting that uses $m \log P$ machines (recall that the adversary uses m machines). They also showed that there exists a $O(\log n)$-machine $O(1)$-speed algorithm that returns the optimal schedule for the unweighted flow-time objective. Epstein et al. [6] proposed an ℓ-machines $O(\min\{\sqrt[\ell]{P}, \sqrt[\ell]{n}\})$-competitive algorithm for the unweighted case on a single machine. This algorithm is optimal up to a constant factor for constant ℓ.

Lucarelli et al. [9] presented a strong lower bound on the competitiveness for the weighted flow-time problem on a single machine that uses arbitrarily faster machine than that of the adversary. Choudhury et al. [5] extended the resource augmentation model to allow *rejection*, according to which the algorithm does not need to complete all jobs and some of them can be rejected. Using a combination of speed augmentation and rejection, Lucarelli et al. [9] gave a constant competitive algorithm for the weighted flow-time problem on a set of unrelated machines. In particular, they showed that there exists a $O(1/\epsilon^2)$-competitive algorithm that uses machines with speed $(1+\epsilon)$ and rejects at most an ϵ-fraction of jobs for arbitrarily small $\epsilon > 0$. Recently, Lucarelli et al. [8] provided a scalable competitive algorithm for the case of (unweighted) flow time where there is no speed augmentation.

2 Definitions and Notations

2.1 Problem definition

We are given a set $\mathcal{M}$ of unrelated machines and a set of jobs $\mathcal{J}$ that arrive online. Each job j is characterized by its release time r_j and its weight w_j. If job j is executed on machine i, it has a processing requirement of p_{ij} time units. The goal is to schedule jobs *non-preemptively*. Given a schedule $\mathcal{S}$, the *completion time* of the job j is denoted by $C_j^{\mathcal{S}}$. The *flow-time* of j is defined as $F_j^{\mathcal{S}} = C_j^{\mathcal{S}} - r_j$, which is the total amount of time job j remains in the system. The objective is to minimize the weighted flow-times of all jobs, i.e., $\sum_{j \in \mathcal{J}} w_j F_j^{\mathcal{S}}$. In the following section we formulate this problem as a linear program.

2.2 Linear Programming Formulation

The LP formulation presented below is an extension of those used in the prior works of [1, 9]. For each job j, machine i and time $t \geq r_j$, there is a binary variable x_{ijt} which indicates if j is processed or not on i at time t. The problem of minimizing weighted flow-time can be expressed as:

$$\min \sum_{i,j,t} w_j \left(\frac{t - r_j}{p_{ij}} + 21 \right) x_{ijt}$$

$$\sum_{i,t} \frac{x_{ijt}}{p_{ij}} = 1 \qquad \forall j \tag{1}$$

$$\sum_{j} x_{ijt} \leq 1 \qquad \forall i, t \tag{2}$$

$$x_{ijt} \in \{0,1\} \qquad \forall i, j, t \geq r_j \tag{3}$$

The objective value of the above integer program is at most a constant factor than that of the optimal preemptive schedule. The above integer program can be relaxed to a linear program by replacing the integrality constraints of x_{ijt} with $0 \leq x_{ijt} \leq 1$. The dual of the relaxed linear program can be expressed as follows:

$$\max \sum_j \alpha_j - \sum_{i,t} \beta_{it}$$

$$\frac{\alpha_j}{p_{ij}} - \beta_{it} \leq w_j \left(\frac{t - r_j}{p_{ij}} + 21 \right) \qquad \forall i, j, t \geq r_j \tag{4}$$

$$\beta_{it} \geq 0 \qquad \forall i, t \tag{5}$$

For the *rejection model* considered in this work, it is assumed that the algorithm is allowed to reject jobs. Rejection can be interpreted in the primal LP by only considering constraints corresponding to non-rejected jobs. That is, the algorithm does not have to satisfy the constraint (1) for rejected jobs.

2.3 Notations

In this section, we define notations that will be helpful during the design and analysis of the algorithm.

- t^- denotes the time just before t that is, $t^- = t - \epsilon'$ for an arbitrarily small value of $\epsilon' > 0$.
- $U_i(t)$ denotes the set of pending jobs at time t on machine i, i.e., the set of jobs dispatched to i that have not yet completed and also have not been rejected until t.
- $\kappa_i(t)$ denotes the job currently executing on machine i at time t.
- $V_i(t)$ denotes the set of unprocessed jobs in $U_i(t)$ that is $V_i(t) = U_i(t) \backslash \{\kappa_i(t)\}$. Throughout this paper, we assume that the jobs in $V_i(t)$ are indexed in non-increasing order of their densities that $\delta_{i1} \geq \delta_{i2}, \ldots, \geq \delta_{i|V_i(t)|}$.
- $\nu_i(t)$ denotes the smallest density job in $V_i(t)$.
- $R_i^1(a, b)$ denotes the set of jobs rejected due to the *prempt* rule (to be defined later) during time interval $(a, b]$. In particular, $R_i^1(t)$ is the set of job rejected at time t due to the *prempt rule.*
- Similarly, $R_i^2(a, b)$ denotes the set of jobs rejected due to the *weight-gap* rule (also to be defined later) during time interval $(a, b]$. In particular, $R_i^2(t)$ is the set of job rejected at time t due to the weight-gap rule.
- $q_{ij}(t)$ denotes the remaining processing time of j at a time t on machine i.
- δ_{ij} is the density of a job j on machine i that is $\delta_{ij} = \frac{w_i}{p_{ij}}$.
- S_j denotes the starting of job j on some machine i. If a job is rejected before it starts executing, set $S_j = \infty$.

By the previous definitions, it follows that $R_i^1(r_j), R_i^2(r_j) \subseteq U(r_j^-) \cup \{j\}$ and $U(r_j) = (U(r_j^-) \cup \{j\}) \setminus \{R_i^1(r_j) \cup R_i^2(r_j)\}$.

3 The Algorithm

In this section, we describe our algorithm. Specifically, we explain how to take the following decisions: *dispatching* that is to decide the machine assignment of jobs; *scheduling* that is to decide which jobs to process at each time; and *rejection*. The algorithm is denoted by $\mathcal{A}$. Let $0 < \epsilon < 1$ be an arbitrarily small constant. Note that the proposed algorithm rejects an $O(\epsilon)$-fraction of the total weight of jobs and dispatches each job to a machine upon its arrival.

3.1 Scheduling policy

At each time t if the machine i is *idle* either due to the rejection of a job or due to the completion of a job, then the algorithm starts executing the job j with the highest density among all the jobs in $U_i(t)$, *i.e.* $j = \arg\max_{h \in U_i(t)} \delta_{ih}$. In case of ties, the algorithm selects the job with the earliest release time.

3.2 Rejection policies

Our algorithm uses two different rules for rejecting jobs. The first rule called as the *preempt rule*, bounds the total weight of "high priority" jobs that arrive during the execution of a job. The second rule called as the *weight-gap* rule, helps the algorithm to balance the total amount of weight of low density jobs. The algorithm associates two counters, count^1_j and count^1_j, with each job j which are both initialized to 0 at r_j.

1. **Preempt rule**: Let $j = \kappa_i(t)$ be the job processing on i at time t. During the processing of j, if a new job j' is dispatched to i then count^1_j is incremented by $w_{j'}$. Let k be the earliest job released and dispatched to machine i during the execution of j such that $\mathsf{count}^1_j \geq w_j/\epsilon$, if it exists. At r_k, the algorithm interrupts the processing of j and rejects it, that is $R^1_i(r_k) = \{j\}$. If no job is rejected due to the *preempt* rule at r_k, then we set $R^1_i(r_k) = \emptyset$.
2. **Weight-gap rule**: We associate a function $W_i(t) : \mathbb{R}^+ \to \mathbb{R}^+$ with each machine i which is initialized to 0 for every t. Informally $W_i(t)$ represents the total budget for future rejections. If a job j is dispatched to machine i then $W_i(t)$ for $t \geq r_j$ is updated according to the following policy.
   Let $V = V_i(r_j^-) \cup \{j\}$. Assume that the jobs in V are indexed in non-increasing order of their densities that is, $\delta_{i1} \geq \delta_{i2} \geq \ldots \geq \delta_{i\nu}$, where the job with index ν is the smallest density job in V. Note that the job j is included in this ordering. Let s be the smallest index in $\{1, 2, \ldots, \nu\}$ such that:

$$\sum_{h=s}^{\nu} w_h \leq \epsilon(W_i(t^-) + w_j) < \sum_{h=(s-1)}^{\nu} w_h \tag{6}$$

   We say that no such job with index s exists if and only if $w_\nu > \epsilon(W_i(t^-)+w_j)$. Algorithm 1 defines the set of jobs $R^2_i(r_j)$. The algorithm rejects the jobs in $R^2_i(r_j)$ and updates $W_i(t)$ as follows:

$$W_i(t) = \max\{0, W_i(r_j^-) + w_j - \sum_{h \in R^2_i(r_j)} w_h/\epsilon\}, \qquad \forall t \geq r_j \tag{7}$$

The following lemma describes some properties arising due to the *weight-gap* rule.

▶ **Lemma 2.** *The following properties hold.*
(Property 1) *If $R^2_i(r_j) = \{\nu_i(r_j^-), j\}$ or $R^2_i(r_j) = \{(s-1), \ldots, v\}$ then $W_i(r_j) = 0$.*
(Property 2) *$\epsilon W_i(t) < w_{\nu_i(t)}$ for every pair of i, t.*
(Property 3) *Let $w_{|R^2_i(r_j)|}$ denote the weight of smallest density job in $R^2_i(r_j)$. If $j \notin R^2_i(r_j)$ then $\sum_{h \in R^2_i(r_j)} w_h - w_{|R^2_i(r_j)|} \leq 2\epsilon w_j$.*
(Property 4) *If $j \in R^2_i(r_j)$, then $R^2_i(r_j) = \{j\}$ or $\{j, \nu_i(r_j^-)\}$.*

▶ **Lemma 3.** *The total weight of jobs rejected by the* preempt rule *is at most $O(\epsilon)$-fraction of the total weight of jobs in $\mathcal{J}$.*

Algorithm 1 Weight-gap Rejection Rule.

```
1: if no job with index $s$ exists then
2:   if $j$ is not the smallest density job in $V$ then
3:     No job is rejected that is, $R_i^2(r_j) := \emptyset$
4:   else
5:     {$j$ is the smallest density job in $V$ that is, $j$ is the job with index $\nu$}
6:     {$v_i(r_j^-)$ is the job with index $\nu - 1$}
7:     if $p_{ij} \geq \epsilon p_{i(\nu-1)}$ then
8:       No job is rejected that is, $R_i^2(r_j) := \emptyset$
9:     else
10:      $\mathsf{count}^2_{(\nu-1)} := \mathsf{count}^2_{(\nu-1)} + w_j$
11:      if $\mathsf{count}^2_{(\nu-1)} \geq w_{(\nu-1)}$ then
12:        Reject $j$ and $\nu_i(r_j^-)$ that is, $R_i^2(r_j) := \{j, \nu_i(r_j^-)\}$
13:      else
14:        No job is rejected that is, $R_i^2(r_j) := \emptyset$
15: else
16:   {a job with index $s$ exists}
17:   if $w_j \geq w_{(s-1)}/\epsilon$ then
18:     Reject jobs with indices $s-1, \ldots, \nu$ in $V$ that is, $R_i^2(r_j) := \{s-1, \ldots, \nu\}$
19:   else
20:     {$w_j < w_{(s-1)}/\epsilon$}
21:     if $j$ is not one of the jobs in $s, \ldots, \nu$ that is, $j \notin \{s, \ldots, \nu\}$ then
22:       Reject jobs with indices $s, \ldots, \nu$ in $V$ that is, $R_i^2(r_j) := \{s, \ldots, \nu\}$
23:     else
24:       {$j \in \{s, \ldots, \nu\}$}
25:       $\mathsf{count}^2_{(s-1)} := \mathsf{count}^2_{(s-1)} + w_j$
26:       if $\mathsf{count}^2_{(s-1)} \geq w_{(s-1)}$ then
27:         Reject jobs with indices $s-1, \ldots, \nu$ in $V$ that is, $R_i^2(r_j) := \{s-1, \ldots, \nu\}$.
28:       else
29:         Reject jobs with indices $s, \ldots, \nu$ in $V$ that is, $R_i^2(r_j) := \{s, \ldots, \nu\}$.
```

Proof. From *preempt* rule, it follows that each job j can be associated with a set of jobs such that their total weight is at most w_j/ϵ. For every pair of j, j' and $j \neq j'$, the intersection of the associated sets is empty and hence the lemma follows. ◀

▶ **Lemma 4.** *The total weight of jobs rejected by the* weight-gap *rule is at most $O(\epsilon)$-fraction of the total weight jobs in $\mathcal{J}$.*

3.3 Dispatching policy

When a new job j arrives, a variable Δ_{ij} is set. Intuitively, Δ_{ij} is the approximate increase in the total weighted flow-time objective if the job j is assigned to the machine i and j is not rejected. Then, Δ_{ij} is defined as follows.

$$\Delta_{ij} = w_j \sum_{h \in V_i(r_j): \delta_{ih} \geq \delta_{ij}} p_{ih} + p_{ij} \sum_{h \in V_i(r_j): \delta_{ih} < \delta_{ij}} w_h$$
$$+ w_j q_{i\kappa_i(r_j^-)}(r_j) \cdot \mathbb{1}_{\{\kappa_i(r_j^-) \text{ is not rejected currently due to preempt rule}\}}$$

$$- q_{i\kappa_i(r_j^-)}(r_j) \cdot \sum_{h \in U_i(r_j)\setminus\{j\}} w_h \cdot \mathbb{1}_{\{\kappa_i(r_j^-) \text{ is currently rejected due to the preempt rule}\}}$$

The first term corresponds to the flow-time of the new job j due to waiting on jobs with higher density than δ_{ij} in $V_i(r_j)$. The second term corresponds to the delay of the jobs in $V_i(r_j)$ with smaller density than δ_{ij}. The third and the fourth terms give corrections depending on whether job $\kappa_i(r_j^-)$ is rejected due to the *preempt* rule.

We now describe the dispatching policy of jobs to machines. At the arrival time of a job j, we hypothetically assign j to every machine i and compute the variables α_{ij}. Finally, we assign j to the machine that minimizes α_{ij}. For notional purposes, we put an additional apostrophe to previously defined variables. The additional apostrophe stands for the fact that these variables correspond to the case where we hypothetically assign j to i. For example, $R_i^{2'}(r_j)$ denote the set of rejected jobs due to the *weight-gap rule* when j is hypothetically assigned to i. Similarly, $W_i'(r_j)$ denote the function W_i at r_j in the case if j is assigned to i. Further, let $\rho = \rho_{ij}$ be an index of a job in $V_i(r_j^-)$ such that the following two inequalities hold simultaneously:

$$\sum_{h=\rho}^{|V_i(r_j^-)|} w_h \le W_i'(r_j) < \sum_{h=(\rho-1)}^{|V_i(r_j^-)|} w_h$$

The variable α_{ij} is computed for each machine i as follows:

$$\alpha_{ij} = \frac{20 w_j p_{ij}}{\epsilon} + w_j \sum_{h \in V_i(r_j^-):\delta_{ih} \ge \delta_{ij}} p_{ih} + w_j p_{ij} + p_{ij} \sum_{h \in V_i(r_j^-):\delta_{ij} > \delta_{ih}} w_{ih} - n_{ij}$$

where n_{ij} is defined as follows.

$$n_{ij} = w_j \left(\sum_{h \in V_i(r_j^-):\delta_{i\rho} \ge \delta_{ih}} p_h + \left(W_i'(r_j) - \sum_{h \in V_i(r_j^-):\delta_{i\rho} \ge \delta_{ih}} w_h \right) \frac{p_{i,(\rho-1)}}{w_{(\rho-1)}} \right) \qquad \text{if } R_i^{2'}(r_j) = \{j\},$$

$$n_{ij} = w_j \sum_{h \in R_i'^2(r_j)} p_{ih} \qquad \text{if } R_i'^2(r_j) = \{j, \nu_i(r_j^-)\},$$

$$n_{ij} = p_{ij} \sum_{h \in R_i'^2(r_j)} w_h + \epsilon^2 W_i'(r_j) p_{ij} \qquad \text{otherwise.}$$

The algorithm assigns j to machine $i^* = \arg\min_{i \in \mathcal{M}} \alpha_{ij}$.

3.4 Dual variables

Suppose job j is assigned to machine i. Assume L_j represents the last time t such that j is in $U_i(t)$. Informally, L_j is the time at which j is removed from the queue of the machine i. Note that j can be removed from $U_i(t)$ due to three following reasons:

1. If j has being scheduled for p_{ij} time units on machine i then $L_j = C_j$
2. If j is rejected due to *preempt* rule
3. If j is rejected due to *weight-gap* rule.

In cases 2 and 3, j is rejected due to the arrival of some job, denoted by $\mathsf{rej}(j)$. Recall that $R_i^1(r_j, L_j)$ is the set of jobs that are rejected due to *preempt rule* during the interval $(r_j, L_j]$ on machine i. Note that those jobs cause a decrease in the flow of j. Observe that $R_i^1(r_j, L_j)$ contains j if j is rejected due to the *preempt* rule. We define the *definitive completion* time, denoted by $\widetilde{C}_j$, of a job j as follows.

1. If j is not rejected due to the *weight-gap* rule (corresponds to cases 1 and 2) .

$$\widetilde{C}_j = L_j + \sum_{h \in R_i^1(r_j, L_j)} q_{ih}(r_{\mathsf{rej}(h)}) \tag{8}$$

2. If j is rejected due to the *weight-gap* rule on the arrival of some job other than j that is, $r_{j'}$ where $j' \neq j$

$$\widetilde{C}_j = L_j + \sum_{h \in R_i^1(r_j, L_j)} q_{ih}(r_{\mathsf{rej}(h)}) + \sum_{h \in U_i(L_j) : \delta_{ih} \geq \delta_{ij}} q_{ih}(L_j) + \sum_{h \in R_i^2(r_{\mathsf{rej}(j)}) : \delta_{ih} \geq \delta_{ij}} p_{ih} \tag{9}$$

3. If j is immediately rejected (i.e., $j \in R_i^2(r_j)$) and job $\nu_i(r_j^-)$ is also rejected due to the arrival of j.

$$\widetilde{C}_j = L_j + p_{ij} + \sum_{h \in U_i(L_j)} q_{ih}(L_j) \tag{10}$$

4. If j is immediately rejected and it is the only job rejected due to the *weight gap* rule at r_j. Denote $\rho = \rho_{ij}$.

$$\begin{aligned}\widetilde{C}_j = L_j + p_{ij} + &\sum_{h \in V_i(L_j^-) : \delta_{ih} > \delta_{i(\rho-1)}} p_{ih} \\ + &\left(1 - \frac{W_i(L_j) - \sum_{h \in V_i(L_j^-) : \delta_{i\rho} \geq \delta_{ih}} w_h}{w_{(\rho-1)}}\right) p_{i(\rho-1)} + q_{i\kappa_i(L_j)}(L_j).\mathbb{1}_{\{R_i^1(L_j) = \emptyset\}}\end{aligned} \tag{11}$$

This completes the description of the *definitive completion time.*

Let $Q_i(t)$ denote the set of jobs that have not been definitely completed that is

$Q_i(t) := \{j : j \text{ has been assigned to } i, t < \widetilde{C}_j\}.$

Next, we define the notion of *artificial fractional weight* of a job $j \in Q_i(t)$,

$$w_j^f(t) = \begin{cases} w_j & \text{if } r_j \leq t \leq \widetilde{C}_j - p_{ij} \\ w_j \left(\frac{\widetilde{C}_j - t}{p_{ij}}\right) & \text{if } \widetilde{C}_j - p_{ij} < t < \widetilde{C}_j \end{cases}$$

Now, we have all the necessary tools to set dual variables. At the arrival of job j, set

$$\alpha_j = \left(\frac{\epsilon}{1+\epsilon}\right) \min_{i \in \mathcal{M}} \alpha_{ij}$$

and never change this value again. The second dual variable β_{it} is set to

$$\frac{\epsilon}{(1+\epsilon)(1+\epsilon^2)} \sum_{h \in Q_i(t)} w_h^f(t)$$

Let $Q_i^R(t) \subseteq Q_i(t)$ be the set of jobs that are rejected due to the *weight-gap rule* and are not yet definitively completed until time t.

▶ **Lemma 5.** *For fixed time t, β_{it} may only increase as new jobs arrive and some old jobs might get rejected.*

Observe that above lemma holds as jobs are removed from $Q_i(t)$ only after their *definitive completion time*. Thus a job that might have already completed its execution on a machine or rejected, can still be present in the $Q_i(t)$. During the analysis, we will show that the dual constraint corresponding to job j are feasible at r_j. Since β_{it} only increases with respect to the arrival of new jobs, the feasibility holds for all $t \geq r_j$.

4 Analysis

We present first two technical lemmas which are important for the analysis of our primal-dual algorithm. In Lemma 6, we relate the weight of rejected jobs in $Q_i^R(t)$ to the weight of jobs pending in $U_i(t)$. This will help us in proving the feasibility of dual constraints in Lemma 9, Lemma 10 and Lemma 11. In Lemma 7, we show that the negative parts in the definition of α_js' are relatively small. This will help us to bound the value of the dual objective.

▶ **Lemma 6.** *Let $\kappa = \kappa_i(t)$. For any machine i and any time t, it holds that $\frac{w_\kappa}{p_{i\kappa}} q_{i\kappa}(t) + \sum_{h \in V_i(t)} w_h^f(t) - W_i(t) \leq \frac{1}{\epsilon} \sum_{h \in Q_i^R(t)} w_h(t)$.*

Proof. We prove by induction on the arrival of jobs. The base case when no job has been released, holds trivially. Assume that the above inequality holds for every time t, before the arrival of job j on machine i, we show that it holds after j arrives. We split the proof into two cases depending upon if j is immediately rejected or not. The rest of the proof is omitted due to space constraints. ◀

▶ **Lemma 7.** *Let $J_i(t)$ denote the set of jobs dispatched to machine i until the time t that is, $J(i) = \bigcup_{t' \leq t} U_i(t')$. Then the following inequality holds at all time and for all $i \in \mathcal{M}$*

$$\mathcal{D}^1 - \mathcal{D}^2 \leq \mathcal{B}^1 + \mathcal{B}^2 + \mathcal{B}^3 \tag{12}$$

where

$$\mathcal{D}^1 = \sum_{j \in J_i(t) \setminus R_i^2(r_j)} \left(\epsilon^2 W_i(r_j) p_{ij} - w_j p_{i,\nu_i(r_j^-)} \cdot \mathbb{1}_{\{j=\nu_i(r_j) \textit{ and } p_j < \epsilon p_{i\nu_i(r_j^-)}\}} \right),$$

$$\mathcal{D}^2 = \sum_{j \in R_i^2(r_j)} \left(w_j p_{i,\nu_i(r_j)} \cdot \mathbb{1}_{\{|R_i^2(r_j)|=1\}} + w_{\nu_i(r_j^-)} p_{i,\nu_i(r_j^-)} \cdot \mathbb{1}_{\{|R_i^2(r_j)|>1\}} \right),$$

$$\mathcal{B}^1 = \sum_{j \in R_i^2(0,t)} w_j p_{ij}, \quad \mathcal{B}^2 = \sum_{j \in J_i(t) \setminus \{R_i^2(0,t) \cup U_i(t)\}} w_j p_{ij} + \epsilon W_i(t) p_{i,\nu_i(t)} \textit{ and}$$

$$\mathcal{B}^3 = \sum_{j \in J_i(t)} w_j p_{ij} / \epsilon.$$

Proof. The proof is omitted due to space constraints. ◀

▶ **Corollary 8.** *Let $J_i \subseteq \mathcal{J}$ be the set of jobs dispatched to machine i that $J_i = \bigcup_{t \geq 0} U_i(t)$. Then the following inequality holds for every machine $i \in \mathcal{M}$,*

$$\sum_{j \in \mathcal{J}_i \setminus R_i^2(r_j)} \epsilon^2 W_i(r_j) p_{ij} \leq \left(\frac{5}{\epsilon} \right) \sum_{j \in \mathcal{J}_i} w_j p_{ij}$$

Proof. From Lemma 7, it immediately follows that

$$\sum_{j \in J_i \setminus R_i^2(r_j)} (\epsilon^2 W_i(r_j) p_{ij} - w_j p_{i,\nu_i(r_j^-)} \cdot \mathbb{1}_{\{j=\nu(r_j) \text{ and } p_j < \epsilon p_{i,\nu_i(r_j^-)}\}})$$
$$- \sum_{j \in R_i^2(r_j)} \left(w_j p_{i,\nu_i(r_j)} \cdot \mathbb{1}_{\{|R_i^2(r_j)|=1\}} + w_{\nu_i(r_j^-)} p_{i,\nu_i(r_j^-)} \cdot \mathbb{1}_{\{|R_i^2(r_j)|>1\}} \right) \leq \frac{2}{\epsilon} \left(\sum_{j \in J_i} w_j p_{ij} \right)$$

Rearranging the terms, we get

$$\epsilon^2 \sum_{j \in J_i \setminus R_i^2(r_j)} W_i(r_j) p_{ij}$$
$$- \sum_{j \in R_i^2(r_j)} \left(w_j p_{i,\nu_i(r_j)} \cdot \mathbb{1}_{\{|R_i^2(r_j)|=1\}} + w_{\nu_i(r_j^-)} p_{i,\nu_i(r_j^-)} \cdot \mathbb{1}_{\{|R_i^2(r_j)|>1\}} \right)$$
$$\leq \frac{2}{\epsilon} \left(\sum_{j \in J_i} w_j p_{ij} \right) + \sum_{j \in J_i \setminus R_i^2(r_j)} w_j p_{i,v_i(r_j^-)} \cdot \mathbb{1}_{\{j=v(r_j) \text{ and } p_j < \epsilon p_{iv_i(r_j^-)}\}}$$
$$\leq \frac{2}{\epsilon} \left(\sum_{j \in J_i} w_j p_{ij} \right) + \sum_{h \in J_i} p_{ih} \sum_{j \in J_i : j=\nu(r_j), h=\nu(r_j^-)} w_j$$
$$\leq \frac{2}{\epsilon} \left(\sum_{j \in J_i} w_j p_{ij} \right) + \sum_{h \in J_i} p_{ih} w_h / \epsilon$$

since $\mathsf{count}_h^2 < w_h/\epsilon$, otherwise h is rejected due to Line 12 in Algorithm 1

$$\leq \frac{3}{\epsilon} \left(\sum_{j \in J_i} w_j p_{ij} \right)$$

Rearranging the terms again, we get

$$\epsilon^2 \sum_{j \in J_i \setminus R_i^2(r_j)} W_i(r_j) p_{ij}$$
$$\leq \frac{3}{\epsilon} \left(\sum_{j \in J_i} w_j p_j \right) + \sum_{j \in R_i^2(r_j)} \left(w_j p_{i,\nu_i(r_j)} \cdot \mathbb{1}_{\{|R_i^2(r_j)|=1\}} + w_{\nu_i(r_j^-)} p_{i,\nu_i(r_j^-)} \cdot \mathbb{1}_{\{|R_i^2(r_j)|>1\}} \right)$$
$$\leq \frac{4}{\epsilon} \left(\sum_{j \in J_i} w_j p_j \right) + \sum_{j \in R_i^2(r_j)} \left(w_j p_{i,\nu_i(r_j)} \cdot \mathbb{1}_{\{|R_i^2(r_j)|=1\}} \right) \quad \text{since } R_i^2(r_j) = \{j, \nu_i(r_j^-)\}$$
$$\leq \frac{4}{\epsilon} \left(\sum_{j \in J_i} w_j p_j \right) + \sum_{h \in J_i} p_{ih} \sum_{j \in J_i : h=\nu_i(r_j^-)=\nu_i(r_j)} w_j$$
$$\leq \frac{5}{\epsilon} \left(\sum_{j \in J_i} w_j p_j \right)$$

The last inequality holds since $\mathsf{count}_h^2 < w_h/\epsilon$, otherwise h is rejected in Line 27 in Algorithm 1. Thus, the corollary follows. ◀

Now we show the proof of dual feasibility for each job j on every pair of i, t. Thus, for a given machine i, j may or may not be assigned to i. by the algorithm.

▶ **Lemma 9.** *Suppose that a job j is not immediately rejected at r_j when j is hypothetically assigned to i. Then, the dual constraint (4) corresponding to j holds.*

▶ **Lemma 10.** *Assume that a job j is immediately rejected at r_j and $R_i^2(r_j) = \{j, \nu_i(r_j^-)\}$ when j is hypothetically assigned to i. Then, the dual constraint (4) corresponding to j holds.*

▶ **Lemma 11.** *Suppose that a job j is immediately rejected at r_j and $R_i^2(r_j) = \{j\}$ when j is hypothetically assigned to i. Then, the dual constraint (4) corresponding to j holds.*

▶ **Lemma 12.** *It holds that $\sum\limits_{j \in \mathcal{J}} \alpha_j \geq \frac{\epsilon}{1+\epsilon} \sum\limits_{j \in \mathcal{J}} (\widetilde{C}_j - r_j)$.*

The proofs of above lemmas are omitted due to space constraints

4.1 Proof of theorem 1

Proof. In the definition of dual variables, each job j is accounted in β_{it} variable until its *definitive completion time*. Thus, $\sum\limits_{i,t} \beta_{it} \leq \frac{\epsilon}{(1+\epsilon)(1+\epsilon^2)} \sum_{j \in \mathcal{J}} w_j(\widetilde{C}_j - r_j)$. Combining it with Lemma 12, we have that the dual objective is at least $\frac{\epsilon^3}{(1+\epsilon)(1+\epsilon^2)} \sum_{j \in \mathcal{J}} w_j(\widetilde{C}_j - r_j)$. Further, the cost of the primal is at most $22 \sum_{j \in \mathcal{J}} w_j(\widetilde{C}_j - r_j)$. Hence the theorem follows. ◀

References

1 S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *Proceedings of Symposium on Discrete Algorithms (SODA, 2012)*, pages 1228–1241, 2012.

2 Nikhil Bansal and Ho-Leung Chan. Weighted flow time does not admit o(1)-competitive algorithms. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA, 2009)*, pages 1238–1244, 2009.

3 Nikhil Bansal and Kedar Dhamdhere. Minimizing weighted flow time. *ACM Transactions on Algorithms*, 3(4), 2007.

4 Chandra Chekuri, Sanjeev Khanna, and An Zhu. Algorithms for minimizing weighted flow time. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 84–93, 2001.

5 Anamitra Roy Choudhury, Syamantak Das, Naveen Garg, and Amit Kumar. Rejecting jobs to minimize load and maximum flow-time. In *Proc. Symposium on Discrete Algorithms*, pages 1114–1133, 2015.

6 Leah Epstein and Rob van Stee. Optimal on-line flow time with resource augmentation. *Discrete Applied Mathematics*, 154(4):611–621, 2006.

7 Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

8 Giorgio Lucarelli, Benjamin Moseley, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online non-preemptive scheduling on unrelated machines with rejections. In *ACM Symposium on Parellelism in Algorithms and Architectures (SPAA, 2018)*, page To appear, 2018.

9 Giorgio Lucarelli, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online non-preemptive scheduling in a resource augmentation model based on duality. In *European Symposium on Algorithms (ESA, 2016)*, volume 57, pages 1–17, 2016.

10 Cynthia A Phillips, Clifford Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

Finding Stable Matchings That Are Robust to Errors in the Input

Tung Mai
Georgia Institute of Technology, Atlanta, GA, USA
tung.mai@cc.gatech.edu

Vijay V. Vazirani
University of California, Irvine, Irvine, CA, USA
vazirani@ics.uci.edu

Abstract

In this paper, we introduce the issue of finding solutions to the stable matching problem that are robust to errors in the input and we obtain the first algorithmic results on this topic. In the process, we also initiate work on a new structural question concerning the stable matching problem, namely finding relationships between the lattices of solutions of two "nearby" instances.

Our main algorithmic result is the following: We identify a polynomially large class of errors, D, that can be introduced in a stable matching instance. Given an instance A of stable matching, let B be the instance that results after introducing one error from D, chosen via a discrete probability distribution. The problem is to find a stable matching for A that maximizes the probability of being stable for B as well. Via new structural properties of the type described in the question stated above, we give a polynomial time algorithm for this problem.

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

Keywords and phrases Stable Matching, Robust

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.60

Funding Supported in part by NSF grant CCF-1815901.

Acknowledgements We wish to thank David Eppstein, Mike Goodrich and Sandy Irani for interesting discussions that sparked off the question addressed in this paper.

1 Introduction

Ever since its introduction in the seminal 1962 paper of Gale and Shapley [3], the stable matching problem has been the subject of intense study from numerous different angles in many fields, including computer science, mathematics, operations research, economics and game theory, e.g., see the books [9, 6, 11]. The very first matching-based market, namely matching medical interns to hospitals, was built around this problem, e.g., see [6, 12]. Eventually, this led to an entire inter-disciplinary field, namely matching and market design [12]. The stable matching problem and market design were the subject of the 2012 Nobel Prize in Economics, awarded to Roth and Shapley [13].

The current paper initiates the study of this problem from yet another angle, namely robustness to errors in the input. To the best of our knowledge, this issue has not been studied in the context of this problem (see also Section 1.2) even though the design of algorithms that produce robust solutions is already a very well established field, especially as pertaining to robust optimization, e.g., see the books [2, 1].

© Tung Mai and Vijay V. Vazirani;
licensed under Creative Commons License CC-BY

A particularly impressive aspect of the stable matching problem is its deep and pristine combinatorial structure. This in turn has led to efficient algorithms for numerous questions studied about this problem, e.g., see the books mentioned above. A second major contribution of our paper is initiating work on a new structural question, namely finding relationships between the lattices of solutions of two "nearby" instances. In the current paper, and our followup work [10], we restrict ourselves to "nearby" instances which differ in only one agent's preference list. Clearly, this is only the tip of the iceberg as far as "nearby" instances go. Moreover, the structural results are so clean and extensive that they are likely to find algorithmic applications beyond the problem of finding robust solutions. In particular, with ever more interesting matching-based markets being designed and launched on the Internet [12], these new structural properties could find interesting applications and are worth studying further.

We will introduce the problem of finding robust stable matchings via the following model: Alice has an instance A of the stable matching problem, over n boys and n girls, which she sends it to Bob over a channel that can introduce errors. Let B denote the instance received by Bob. Let D denote a polynomial sized domain from which errors are introduced by the channel; we will assume that the channel introduces at most one error from D. We are also given the discrete probability distribution, p over D, from which the channel picks one error. In addition, Alice sends to Bob a matching, M, of her choice, that is stable for instance A. Since M consists of only $O(n)$ numbers of $O(\log n)$ bits each, as opposed to A which requires $O(n^2)$ numbers, Alice is able to send it over an error-free channel. Now Alice wants to pick M in such a way that it has the highest probability of being stable in the instance received by Bob. Hence she picks M from the set

$$\arg\max_N \{Pr_p[N \text{ is stable for instance } B \mid N \text{ is stable for instance } A]\},$$

We will say that such a matching M is *robust.* We seek a polynomial time algorithm for finding such a matching.

Clearly, the domain of errors, D, will have to be well chosen to solve this problem. A natural set of errors is *simple swaps*, under which the positions of two adjacent boys in a girl's list, or two adjacent girls in a boy's list, are interchanged. We will consider a generalization of this class of errors, which we call *shift.* For a girl g, assume her preference list in instance A is $\{\ldots, b_1, b_2, \ldots, b_k, b, \ldots\}$. Move up the position of b so g's list becomes $\{\ldots, b, b_1, b_2, \ldots, b_k, \ldots\}$, and let B denote the resulting instance. Then we will say that B is obtained from A by a shift. An analogous operation is defined on a boy b's list. The domain D consists of all such shifts; clearly, D is polynomially bounded. We prove the following theorem.

▶ **Theorem 1.** *There is a polynomial time algorithm which given an instance A of the stable matching problem and a probability distribution p over the domain, D, of errors defined above, finds a robust stable matching in A.*

1.1 Overview of results and technical ideas

We first summarize some well-known structural facts, e.g., see [6]. The set of stable matchings of an instance form a distributive lattice: given two stable matchings M and M', their meet and join involve taking, for each boy, the optimal or pessimal choice, respectively. It is easy to show that the resulting two matchings are also stable. The extreme matchings of this lattice are called *boy optimal* and *girl-optimal* matchings. A deep notion about this lattice is that of a rotation. A *rotation*, on an ordered list of k boy-girl pairs, when applied to a matching M in which all these boy-girl pairs are matched to each other, matches each boy to the next girl on the list, closing the list under rotation. The k pairs and the order among them are so chosen that the resulting matching is also stable; moreover, a rotation on a subset of these k pairs, under any ordering, leads to a matching that is not stable. Hence, a rotation can be viewed as a minimal change to the current matching that results in a stable matching. Rotations help traverse the lattice from the boy-optimal to the girl-optimal matching along all possible paths available.

For the given instance, a partial order Π is defined on a subset of rotations; the closed sets of Π are in one-to-one correspondence with the set of stable matchings of the instance. Moreover, if S is such a closed set, then starting in the lattice from the boy-optimal matching and applying the rotations in set S, we reach the stable matching corresponding to S.

Let A and B be two instances of stable matching over n boys and n girls, with sets of stable matchings $\mathcal{M}_A$ and $\mathcal{M}_B$, respectively, and lattices $\mathcal{L}_A$ and $\mathcal{L}_B$, respectively. Then, it is easy to see that the matchings in $\mathcal{M}_A \cap \mathcal{M}_B$ form a sublattice in each of the two lattices. Next assume that instance B results from applying a shift operation, defined above, to instance A. Then, we show that $\mathcal{M}_{AB} = \mathcal{M}_A \setminus \mathcal{M}_B$ is also a sublattice of $\mathcal{L}_A$. We use this fact crucially to show that there is at most one rotation, ρ_{in}, that leads from $\mathcal{M}_A \cap \mathcal{M}_B$ to $\mathcal{M}_{AB}$ and at most one rotation, ρ_{out} that leads from $\mathcal{M}_{AB}$ to $\mathcal{M}_A \cap \mathcal{M}_B$. Moreover, we can obtain efficiently this pair of rotations for each of the polynomially many instances that result from the polynomially many shifts.

It is easy to see that a matching M corresponding to a closed set S is stable in instance B iff whenever $\rho_{\text{in}} \in S$, $\rho_{\text{out}} \in S$. We next give an integer program whose optimal solution is a robust stable matching for the given probability distribution on shifts. The IP has one indicator variable, y_ρ, corresponding to each rotation ρ in Π. The constraints of the program ensure that the set S of rotations that are set to 0 form a closed set. The rest of the constraints and the objective function ensure that the corresponding matching maximizes the probability that it is stable in the erroneous instance B. Finally, we show that the LP-relaxation of this IP always has integral solutions. Hence we obtain a polynomial time algorithm for finding a robust stable matching.

1.2 A matter of nomenclature

Assigning correct nomenclature to a new issue under investigation is clearly critical for ease of comprehension. In this context we wish to mention that very recently, Genc et. al. [4] defined the notion of an (a, b)-supermatch as follows: this is a stable matching in which if any a pairs break up, then it is possible to match them all off by changing the partners of at most b other pairs, so the resulting matching is also stable. They showed that it is NP-hard to decide if there is an (a, b)-supermatch. They also gave a polynomial time algorithm for a very restricted version of this problem, namely given a stable matching and a number b, decide if it is a $(1, b)$-supermatch. Observe that since the given instance may have exponentially many stable matchings, this does not yield a polynomial time algorithm even for deciding if there is a stable matching which is a $(1, b)$-supermatch for a given b.

Genc. et. al. [5] also went on to defining the notion of the most robust stable matching, namely a $(1, b)$-supermatch where b is minimum. We would like to point out that “robust” is a misnomer in this situation and that the name “fault-tolerant” is more appropriate. In the literature, the latter is used to describe a system which continues to operate even in the event of failures and the former is used to describe a system which is able to cope with erroneous inputs, e.g., see the following pages from Wikipedia [15, 14].

2 Preliminaries

2.1 The stable matching problem

The stable matching problem takes as input a set of boys $B = \{b_1, b_2, \ldots, b_n\}$ and a set of girls $G = \{g_1, g_2, \ldots, g_n\}$; each person has a complete preference ranking over the set of opposite sex. The notation $b_i <_g b_j$ indicates that girl g strictly prefers b_j to b_i in her preference list. Similarly, $g_i <_b g_j$ indicates that the boy b strictly prefers g_j to g_i in his list.

A matching M is a one-to-one correspondence between B and G. For each pair $bg \in M$, b is called the partner of g in M (or M-partner) and vice versa. For a matching M, we say that b is *above* (or *below*) g if he prefers his M-partner to g (or g to his M-partner). Similarly, g is said to be *above* (or *below*) b if she prefers her M-partner to b (or b to her M-partner). For a matching M, a pair $bg \notin M$ is said to be *blocking* if b is below g and g is below b, i.e., they prefer each other to their partners. A matching M is *stable* if there is no blocking pair in M.

2.2 The lattice of stable matchings

Let M and M' be two stable matchings. We say that M *dominates* M', denoted by $M \preceq M'$, if every boy weakly prefers his partner in M to M'. It is well known that the dominance partial order over the set of stable matchings forms a distributive lattice [6], with meet and join defined as follows. The *meet* of M and M', $M \wedge M'$, is defined to be the matching that results when each boy chooses his more preferred partner from M and M'; it is easy to show that this matching is also stable. The *join* of M and M', $M \vee M'$, is defined to be the matching that results when each boy chooses his less preferred partner from M and M'; this matching is also stable. These operations distribute, i.e., given three stable matchings M, M', M'',

$$M \vee (M' \wedge M'') = (M \vee M') \wedge (M \vee M'') \quad \text{and} \quad M \wedge (M' \vee M'') = (M \wedge M') \vee (M \wedge M'').$$

It is easy to see that the lattice must contain a matching, M_0, that dominates all others and a matching M_z that is dominated by all others. M_0 is called the *boy-optimal matching*, since in it, each boy is matched to his most favorite girl among all stable matchings. This is also the *girl-pessimal matching*. Similarly, M_z is the *boy-pessimal* or *girl-optimal matching*.

2.3 Rotations help traverse the lattice

A crucial ingredient needed to understand the structure of stable matchings is the notion of a rotation, which was defined by Irving [7] and studied in detail in [8]. A rotation takes r matched pairs in a fixed order, say $\{b_0g_0, b_1g_1, \ldots, b_{r-1}g_{r-1}\}$ and "cyclically" changes the mates of these $2r$ agents, as defined below, to arrive at another stable matching. Furthermore, it represents a minimal set of pairings with this property, i.e, if a cyclic change is applied on any subset of these r pairs, with any ordering, then the resulting matching has a blocking pair and is not stable. After rotation, the boys' mates weakly worsen and the girls' mates weakly improve. Thus one can traverse from M_0 to M_z by applying a suitable sequence of rotations (specified by the rotation poset defined below). Indeed, this is precisely the purpose of rotations.

Let M be a stable matching. For a boy b let $s_M(b)$ denote the first girl g on b's list such that g strictly prefers b to her M-partner. Let $next_M(b)$ denote the partner in M of girl $s_M(b)$. A *rotation ρ exposed* in M is an ordered list of pairs $\{b_0g_0, b_1g_1, \ldots, b_{r-1}g_{r-1}\}$ such that for each i, $0 \le i \le r-1$, b_{i+1} is $next_M(b_i)$, where $i+1$ is taken modulo r. In this paper, we assume that the subscript is taken modulo r whenever we mention a rotation. Notice that a rotation is cyclic and the sequence of pairs can be rotated. M/ρ is defined to be a matching in which each boy not in a pair of ρ stays matched to the same girl and each boy b_i in ρ is matched to $g_{i+1} = s_M(b_i)$. It can be proven that M/ρ is also a stable matching. The transformation from M to M/ρ is called the *elimination* of ρ from M.

Let $\rho = \{b_0g_0, b_1g_1, \ldots, b_{r-1}g_{r-1}\}$ be a rotation. For $0 \le i \le r-1$, we say that *ρ moves b_i from g_i to g_{i+1}*, and *moves g_i from b_i to b_{i-1}*. If g is either g_i or is strictly between g_i and g_{i+1} in b_i's list, then we say that *ρ moves b_i below g*. Similarly, *ρ moves g_i above b* if b is b_i or between b_i and b_{i-1} in g_i's list.

2.4 The rotation poset

A rotation ρ' is said to precede (or dominate) another rotation ρ, denoted by $\rho' \prec \rho$, if ρ' is eliminated in every sequence of eliminations from M_0 to a stable matching in which ρ is exposed. Thus, the set of rotations forms a partial order via this precedence relationship. The partial order on rotations is called *rotation poset* and denoted by Π.

▶ **Lemma 2** ([6], Lemma 3.2.1). *For any boy b and girl g, there is at most one rotation that moves b to g, b below g, or g above b. Moreover, if ρ_1 moves b to g and ρ_2 moves b from g then $\rho_1 \prec \rho_2$.*

A *closed subset* is a subset of the poset such that if an element is in the subset then all of its predecessors are also included. There is a one-to-one relationship between the stable matchings and the closed subsets of Π. Given a closed subset C, the correponding matching M is found by eliminating the rotations starting from M_0 according to the topological ordering of the elements in the subset. We say that *C generates M*.

▶ **Lemma 3** ([6], Lemma 3.3.2). *Π contains $O(n^2)$ rotations and can be computed in polynomial time.*

▶ **Lemma 4** ([6], Theorem 2.5.4). *Every rotation appears exactly once in any sequence of elimination from M_0 to M_z.*

2.5 The notion of shift

In this paper, we will assume that a girl applies a *shift* to one of her preferences as defined below. We will study the structural properties of the resulting instance B.

Let the preference list of girl g in A be $\{\ldots, b_1, b_2, \ldots, b_k, b, \ldots\}$. In B the preference list of g is $\{\ldots, b, b_1, b_2, \ldots, b_k, \ldots\}$. Moreover, all other preference lists are identical in both A and B. We say that B obtained from A by applying a *shift.* We denote $x <_z^I y$ if z prefers y to x in instance I.

3 Structural Results

3.1 The stable matchings in $\mathcal{M}_A \setminus \mathcal{M}_B$ form a sublattice

Let $\mathcal{M}_A$ and $\mathcal{M}_B$ be the sets of all stable matchings under instance A and B respectively. Let $\mathcal{M}_{AB} = \mathcal{M}_A \setminus \mathcal{M}_B$. In other words, $\mathcal{M}_{AB}$ is the set of stable matchings in A that become unstable in B. In this section we show that $\mathcal{M}_{AB}$ forms a lattice. We first prove a simple observation.

▶ **Lemma 5.** *Let $M \in \mathcal{M}_{AB}$. The only blocking pair of M under instance B is bg.*

Proof. Since $M \notin \mathcal{M}_B$, there must be a blocking pair $xy \notin M$ under B. Assume xy is not bg, we will show that xy must also be a blocking pair in A. Let y' be the partner of x and x' be the partner of y in M. Since xy is a blocking pair in B, $x >^B_y x'$ and $y >^B_x y'$. The preference list of x remain unchanged from A to B, so $y >^A_x y'$. Next, we consider two cases:

- If y is not g, the preference list of y does not change. Therefore, $x >^A_y x'$, and hence, xy is also a blocking pair in A.
- If y is g, for all pairs x, x' such that $x >^B_y x'$ and $x \neq b$, we also have $x >^A_y x'$. Therefore, xy is a blocking pair in A.

This contradicts the fact that M is stable under A. ◀

Recall that $b_1 \geq_g b_2 \geq_g \ldots \geq_g b_k$ are k boys right above b in g's list such that the position of b is shifted up to be above b_1 in B. From Lemma 5, we can then characterize the set $\mathcal{M}_{AB}$.

▶ **Lemma 6.** *$\mathcal{M}_{AB}$ is the set of all stable matchings in A that match g to a partner between b_1 and b_k in g's list, and match b to a partner below g in b's list.*

Proof. Assume M is a stable matching in A that contains $b_i g$ for $1 \leq i \leq k$ and bg' such that $g >_b g'$. In B, g prefers b to b_i, and hence bg is a blocking pair. Therefore, M is not stable under B and $M \in \mathcal{M}_{AB}$.

To prove the other direction, let M be a matching in $\mathcal{M}_{AB}$. By Lemma 5, bg is the only blocking pair of M in B. For that to happen, $p_M(b) <^B_b g$ and $p_M(g) <^B_g b$. We will show that $p_M(g) = b_i$ for $1 \leq i \leq k$. Assume not, then $p_M(g) <^B_g b_k$, and hence, $p_M(g) <^A_g b$. Therefore, bg is a blocking pair in A, which is a contradiction. ◀

Let $\mathcal{L}_A$ be the boy-optimal lattice formed by $\mathcal{M}_A$.

▶ **Theorem 7.** *$\mathcal{M}_{AB}$ forms a sublattice of $\mathcal{L}_A$.*

Proof. Assume $\mathcal{M}_{AB}$ is not empty. Let M_1 and M_2 be two matchings in $\mathcal{M}_{AB}$. By Lemma 6, M_1 and M_2 both match g to a partner between b_1 and b_k in g's list, and match b to a partner below g in b's list. Since $M_1 \wedge M_2$ is the matching resulting from having each boy choose the more preferred partner and each girl choose the least preferred partner, $M_1 \wedge M_2$ also belongs to the set characterized by Lemma 6. A similar argument can be applied to the case of $M_1 \vee M_2$. Therefore $\mathcal{M}_{AB}$ form a sublattice of $\mathcal{L}_A$. ◀

3.2 Rotations going into and out of a sublattice

Let M be a stable matching in $\mathcal{M}_A$ and ρ be a rotation exposed in M with respect to instance A. If $M \notin \mathcal{S}$ and $M/\rho \in \mathcal{S}$ for a set $\mathcal{S}$, we say that ρ *goes into* $\mathcal{S}$. Similarly, if $M \in \mathcal{S}$ and $M/\rho \notin \mathcal{S}$, we say that ρ *goes out of* S. Let the set of all rotations going into $\mathcal{S}$ and out of $\mathcal{S}$ be $I_{\mathcal{S}}$ and $O_{\mathcal{S}}$, respectively.

Let $\{b_{i_1}, \ldots b_{i_l}\}$ be the set of possible partners of g in any stable matching in $\mathcal{M}_{AB}$, where $1 \leq i_1 \leq \ldots \leq i_l \leq k$. Let ρ_1 be a rotation moving g to b_{i_l}, ρ_2 be the rotation moving b below g and ρ_3 be a rotation moving g from b_{i_1}. Note that each of ρ_1, ρ_2 and ρ_3 might not exist.

▶ **Lemma 8.** *$I_{\mathcal{M}_{AB}}$ can only contain ρ_1, ρ_2. $O_{\mathcal{M}_{AB}}$ can only contain ρ_3.*

Proof. Consider a rotation $\rho \in I_{\mathcal{M}_{AB}}$. There exists $M \in \mathcal{M}_A \backslash \mathcal{M}_{AB}$ such that $M/\rho \in \mathcal{M}_{AB}$. By Lemma 6, M/ρ matches g to a partner between b_1 and b_k in g's list, and matches b to a partner below g in b's list. Moreover, M either does not contain b_ig for any $1 \leq i \leq k$, or contains bg' where $g' \geq_b g$, or both. If M does not contain b_ig for any $1 \leq i \leq k$, then $\rho = \rho_1$. If M contains bg' where $g' \geq_b g$, then $\rho = \rho_2$.

Consider a rotation $\rho \in O_{\mathcal{M}_{AB}}$. There exists $M \in \mathcal{M}_{AB}$ such that $M/\rho \in \mathcal{M}_A \setminus \mathcal{M}_{AB}$. Again, by Lemma 6, M contains b_ig for $1 \leq i \leq k$ and bg' where $g' <_b g$. Since M dominates M/ρ in the boy optimal lattice, b must prefer g' to his partner in M/ρ. Hence, M/ρ matches b to a partner below g in b's list. Therefore, M/ρ must not contain b_ig for any $1 \leq i \leq k$. It follows that ρ must be ρ_3. ◀

▶ **Lemma 9.** *If both ρ_1 and ρ_2 exist then $\rho_1 \preceq \rho_2$.*

Proof. Assume that $\rho_1 \neq \rho_2$ and there exists a sequence of rotation eliminations, from M_0 to a stable matching M in which ρ_2 is exposed, that does not contain ρ_1. Since ρ_2 moves b below g, g is matched a partner higher than b in her list in M/ρ_2. Therefore, the partner can only be b_{i_l} or a boy higher than b_{i_l} in g's list.

Consider any sequence of rotation eliminations from M/ρ to M_z. In the sequence, the position of g's partner can only go higher in her list. Therefore, ρ_1 cannot be exposed in any matching in the sequence. It follows that ρ_1 is not exposed in a sequence of eliminations from M_0 to M_z, which is a contradiction by Lemma 4. ◀

▶ **Theorem 10.** *There is at most one rotation in $I_{\mathcal{M}_{AB}}$ and at most one rotation in $O_{\mathcal{M}_{AB}}$. Moreover, the rotation in $I_{\mathcal{M}_{AB}}$ must be either ρ_1 or ρ_2, and the rotation in $O_{\mathcal{M}_{AB}}$ must be ρ_3.*

Proof. By Lemma 8, $I_{\mathcal{M}_{AB}}$ can contain at most 2 rotations, namely ρ_1 and ρ_2 if they are distinct. By Lemma 9, if both of them exist, $\rho_1 \preceq \rho_2$. Hence, $I_{\mathcal{M}_{AB}}$ can contain at most one rotation, and it is either ρ_1 or ρ_2.

Again, by Lemma 8, $O_{\mathcal{M}_{AB}}$ can contain at most one rotation, namely ρ_3 if it exists. ◀

By Theorem 10, there is at most one rotation ρ_{in} coming into $\mathcal{M}_{AB}$ and at most one rotation ρ_{out} coming out of $\mathcal{M}_{AB}$.

▶ **Proposition 11.** *ρ_{in} and ρ_{out} can be computed in polynomial time.*

Proof. Since we can compute Π_A efficiently according to Lemma 3, each of ρ_1, ρ_2 and ρ_3 can be computed efficiently.

First we can check possible partners of b and g with respect to instance A. By Lemma 6, $\mathcal{M}_{AB}$ is empty if none of the possible partners of g is between b_1 and b_k in g's list or none of the partners of b is below g in b's list. It follows that both ρ_{in} and ρ_{out} do not exist. Hence we may assume that such a case does not happen.

Suppose ρ_2 exists. If ρ_3 exists and $\rho_3 \preceq \rho_2$, $\mathcal{M}_{AB} = \emptyset$. Otherwise, $\rho_{\text{in}} = \rho_2$, and $\rho_{\text{out}} = \rho_3$ if ρ_3 exists.

Suppose ρ_2 does not exist. If ρ_1 exists, $\rho_{\text{in}} = \rho_1$. If ρ_3 exists, $\rho_{\text{out}} = \rho_3$. ◀

▶ **Lemma 12.** *Let M be a matching in $\mathcal{M}_{AB}$ and S be the corresponding closed subset in Π_A. If ρ_1 exists, S must contain ρ_1. If ρ_2 exists, S must contain ρ_2. If ρ_3 exists, S must not contain ρ_3.*

Proof. If ρ_1 exists, M_0 does not contain $b_i g$ for any $i \in [1,k]$. Since $M \in \mathcal{M}_{AB}$, by Lemma 6 M matches g to a boy between b_1 and b_k in her list. The set of rotations eliminated from M_0 to M must include ρ_1.

If ρ_2 exists, b can not be below g in M_0. Since b is below g in M, by Lemma 6 the set of rotations eliminated from M_0 to M must include ρ_2.

Assume that ρ_3 exists and S contains ρ_3. Since ρ_3 moves g up from b_{i_1}, M can not contain $b_i g$ for any $i \in [1,k]$. This is a contradiction. ◀

3.3 The rotation poset for the sublattice M_{AB}

From the previous section we know that M_{AB} is a sublattice of M_A. In this section we give the rotation poset that generates all stable matchings in the sublattices.

We may assume that $M_{AB} \neq \emptyset$. If ρ_{in} exists, let $\Pi_{\text{in}} = \{\rho \in \Pi_A : \rho \preceq \rho_{\text{in}}\}$ and M_{boy} be the matching generated by Π_{in}. Otherwise, let $M_{\text{boy}} = M_0$. Similarly, let M_{girl} be the matching generated by $\Pi_A \setminus \Pi_{\text{out}}$, where $\Pi_{\text{out}} = \{\rho \in \Pi_A : \rho \succeq \rho_{\text{out}}\}$, if ρ_{out} exists, and $M_{\text{girl}} = M_z$ otherwise.

▶ **Lemma 13.** *M_{boy} is the boy-optimal matching in $\mathcal{M}_{AB}$, and M_{girl} is the girl-optimal matching in $\mathcal{M}_{AB}$.*

Proof. Let M be a matching in $\mathcal{M}_{AB}$ generated by a closed subset $S \subseteq \Pi_A$. By Lemma 12, if ρ_{in} exists, S must contain ρ_{in}. Since Π_{in} is the minimum set containing ρ_{in}, $\Pi_{\text{in}} \subseteq S$. Therefore, $M_{\text{boy}} \preceq M$.

To prove that $M \preceq M_{\text{girl}}$, we show $S \subseteq \Pi_A \setminus \Pi_{\text{out}}$. Assume otherwise, then there exists a rotation $\rho \in S$ such that $\rho \notin \Pi_A \setminus \Pi_{\text{out}}$. It follows that $\rho \in \Pi_{\text{out}}$, and hence $\rho \succeq \rho_{\text{out}}$. Since S contains ρ and S is a closed subset, S must also contain ρ_{out}. This is a contradiction by Lemma 12. ◀

▶ **Theorem 14.** $\Pi_{AB} = \Pi_A \setminus (\Pi_{in} \cup \Pi_{out})$ *is the rotation poset generating $\mathcal{M}_{AB}$.*

Proof. Let M be a matching in $\mathcal{M}_{AB}$ generated by a closed subset $S \subseteq \Pi_A$. Let $S' = S \setminus \Pi_{\text{in}}$. We show that S' is a closed subset of Π_{AB} and eliminating the rotations in S' starting from M_{boy} according to the topological ordering of the elements gives M.

First $S' \cap \Pi_{\text{in}} = \emptyset$ trivially. Since $M \in \mathcal{M}_{AB}$, S does not contain ρ_{out} by Lemma 12. Therefore, S' does not contain ρ_{out}, and $S' \cap \Pi_{\text{out}} = \emptyset$. It follows that S' is a closed subset of Π_{AB}.

Next observe that we can eliminate rotations in S from M_0 by eliminating rotations in Π_{in} first and then eliminating rotations in $S \setminus \Pi_{\text{in}}$. This can be done because Π_{in} is a closed subset of Π_A. Since Π_{in} generates M, the lemma follows. ◀

4 Algorithm for finding a robust stable matching

We now use the structural properties described in Section 3 to give a polynomial time algorithm for finding a robust stable matching. Clearly, the results in Section 3 can be reproduced when we make a shift in a boy's list. Recall from Section 1 that given a discrete probability distribution $\mathcal{D}$ on all possible shifts, a robust stable matching is a stable matching $M \in \mathcal{M}_A$ that minimizes the probability that $M \in \mathcal{M}_{AB}$, where $B \sim \mathcal{D}$.

For a shift B, let ρ^B_{in} and ρ^B_{out} be the rotation going into $\mathcal{M}_{AB}$ and out of $\mathcal{M}_{AB}$ respectively. By Proposition 11, ρ^B_{in} and ρ^B_{out} can be computed efficiently for each B.

By Lemma 3, Π_A can be computed in polynomial time. We create two additional vertices, a source s and a sink t. For a shift B, we may ignore the cases where neither ρ^B_{in} nor ρ^B_{out} exist. In those cases, either $\mathcal{M}_A = \mathcal{M}_B$ or $\mathcal{M}_A \cap \mathcal{M}_B = \emptyset$. Hence, assume that such an instance B does not exist, and $\mathcal{M}_{AB}$ is always a proper non-empty subset of $\mathcal{M}_A$. For a shift B such that ρ_{in}^B does not exist, let $\rho_{\text{in}}^B = s$. Similarly, for a shift B such that ρ_{out}^B does not exist, let $\rho_{\text{out}}^B = t$.

Let p_B be the probability that instance B is chosen according to D. Consider the following integer program:

$$\begin{aligned}
\min \quad & \sum_B x_B p_B & & \\
\text{s.t.} \quad & y_{\rho_1} \le y_{\rho_2} & & \forall \rho_1, \rho_2 : \rho_1 \prec \rho_2 \\
& y_t = 1 & & \\
& y_s = 0 & & \\
& x_B \ge y_{\rho_{\text{out}}^B} - y_{\rho_{\text{in}}^B} & & \forall B \\
& x_B \ge 0 & & \forall B \\
& y_\rho \in \{0, 1\} & & \forall \rho \in \Pi_A.
\end{aligned} \tag{IP}$$

▶ **Lemma 15.** *(IP) gives a solution to a robust stable matching.*

Proof. Let $S = \{\rho : y_\rho = 0\}$. The set of constraints:

$$y_{\rho_1} \le y_{\rho_2} \quad \forall \rho_1, \rho_2 : \rho_1 \prec \rho_2$$

guarantees that S is a closed subset.

Notice that $x_B = 1$ if and only if $y_{\rho_{\text{out}}^B} = 1$ and $y_{\rho_{\text{in}}^B} = 0$. This, in turn, happens if and only if the matching generated by S is in $\mathcal{M}_{AB}$.

Therefore, by minimizing $\sum_{e \in E} x_B p_B$, we can find a closed subset that generates a robust stable matching. ◀

▶ **Lemma 16.** *(IP) can be solved in polynomial time.*

Proof. Consider relaxing the constraint $y_\rho \in \{0, 1\}$ to $0 \le y_\rho \le 1$. We show how to round a solution of this natural LP-relaxation of (IP) to have an integral solution of the same objective function. It suffices to just consider $\boldsymbol{y}$ as x_B will always be set to $\max(0, y_{\rho_{\text{out}}^B} - y_{\rho_{\text{in}}^B})$ for any given $\boldsymbol{y}$.

Let $\boldsymbol{y}$ be a fractional optimal solution of the relaxation. Let $1 = a_0 > a_1 > a_2 > \ldots > a_k > a_{k+1} = 0$ be all the possible y-values. Since $\boldsymbol{y}$ is fractional, $k \ge 1$. Denote S_i by the set of all rotations having y-value equal to a_i, where $1 > a_i > 0$.

Let $\mathcal{B}^+$ be the set of instances B such that:

- $x_B = y_{\rho_{\text{out}}^B} - y_{\rho_{\text{in}}^B} > 0$.
- $y_{\rho_{\text{out}}^B} = a_i$.
- $y_{\rho_{\text{in}}^B} \ne a_i$.

Let $\mathcal{B}^-$ be the set of instances B such that:

- $x_B = y_{\rho_{\text{out}}^B} - y_{\rho_{\text{in}}^B} > 0$.
- $y_{\rho_{\text{in}}^B} = a_i$.
- $y_{\rho_{\text{out}}^B} \ne a_i$.

Consider perturbing the y-value of all rotations in S_a by a small amount ϵ:

$$y_\rho \leftarrow y_\rho + \epsilon = a_i + \epsilon \quad \forall \rho \in S_a.$$

Here ϵ is chosen so that $a_i + \epsilon < a_{i-1}$ and $a_i + \epsilon > a_{i+1}$. The net change in the objective function is

$$\sum_{B \in \mathcal{B}^+} \epsilon p_B - \sum_{B \in \mathcal{B}^-} \epsilon p_B = \epsilon \left(\sum_{B \in \mathcal{B}^+} p_B - \sum_{B \in \mathcal{B}^-} p_B \right).$$

We claim that

$$\sum_{B \in \mathcal{B}^+} p_B - \sum_{B \in \mathcal{B}^-} p_B = 0.$$

Assume otherwise, we can pick a sign of ϵ to have a strictly smaller objective function. Since $\sum_{B \in \mathcal{B}^+} p_B - \sum_{B \in \mathcal{B}^-} p_B = 0$, we can choose $\epsilon = a_{i-1} - a_i$ and obtain another optimal solution where the value of k decreases by 1. Keep going until $k = 0$ gives an integral solution. ◀

Finally, Theorem 1 follows from Lemmas 15 and 16.

5 Discussion

As stated in the Introduction, the two main questions on stable matching introduced in this paper are obtaining efficient algorithms for finding solutions that are robust to errors in the input, and the structural question of finding relationships between the lattices of solutions of two "nearby" instances. The current paper and our followup work [10] seem to suggest that both these issues are likely to lead to much work in the future. In particular, the structural results are so clean and extensive that they are likely to find algorithmic applications beyond the problem of finding robust solutions. One possible domain of applications that may be able to exploit these structural properties is matching-based markets, particularly as we are seeing ever more interesting such markets being designed and launched on the Internet, e.g., see [12].

At a more detailed level, the domain D, for which we have obtained our algorithm, is very restrictive and we need to extend it to a larger domain. Our followup paper [10] does this, though it seems more can be done. In particular, are there ways of dealing with two or more errors? Another interesting question is to improve the running time of our algorithm. This looks also quite plausible.

Beyond these questions, pertaining to the most basic of formulations of stable matching, one can study numerous variants and generalizations, such as incomplete preference lists, the stable roommates problem, and matching intern couples to hospitals. Each of these bring their own structural properties and challenges, e.g., see [6, 11].

References

1 A. Ben-Tal, L. El Ghaoui, and A.S. Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, October 2009.

2 G. C. Calafiore and L. El Ghaoui. On distributionally robust chance-constrained linear programs. *Journal of Optimization Theory and Applications*, 130(1), 2006.

3 David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

4 Begum Genc, Mohamed Siala, Barry O'Sullivan, and Gilles Simonin. Finding robust solutions to stable marriage. *arXiv preprint arXiv:1705.09218*, 2017.

5 Begum Genc, Mohamed Siala, Gilles Simonin, and Barry O'Sullivan. On the complexity of robust stable marriage. In *International Conference on Combinatorial Optimization and Applications*, pages 441–448. Springer, 2017.

6 Dan Gusfield and Robert W Irving. *The stable marriage problem: structure and algorithms.* MIT press, 1989.

7 Robert W Irving. An efficient algorithm for the "stable roommates" problem. *Journal of Algorithms*, 6(4):577–595, 1985.

8 Robert W Irving and Paul Leather. The complexity of counting stable marriages. *SIAM Journal on Computing*, 15(3):655–667, 1986.

9 Donald Ervin Knuth. *Stable marriage and its relation to other combinatorial problems: An introduction to the mathematical analysis of algorithms.* American Mathematical Soc., 1997.

10 Tung Mai and Vijay V. Vazirani. A generalization of Birkhoff's theorem for distributive lattices, with applications to robust stable matchings. In arXiv, 2018.

11 David Manlove. *Algorithmics of Matching Under Preferences.* World Scientific, 2013.

12 Alvin E. Roth. Al Roth's game theory, experimental economics, and market design page, 2016. URL: `http://stanford.edu/~alroth/alroth.html#MarketDesign`.

13 Alvin E. Roth and Lloyd S. Shapley. The 2012 Nobel Prize in Economics, 2012. URL: `https://www.nobelprize.org/nobel_prizes/economic-sciences/laureates/2012/`.

14 Wikipedia. Fault Tolerence. URL: `https://en.wikipedia.org/wiki/Fault_tolerance`.

15 Wikipedia. Robustness (Computer Science). URL: `https://en.wikipedia.org/wiki/Robustness_(computer_science)`.

Disconnected Cuts in Claw-free Graphs

Barnaby Martin
Department of Computer Science, Durham University, Durham, UK
barnaby.d.martin@durham.ac.uk

Daniël Paulusma
Department of Computer Science, Durham University, Durham, UK
daniel.paulusma@durham.ac.uk

Erik Jan van Leeuwen
Department of Information and Computing Sciences, Utrecht University, The Netherlands
e.j.vanleeuwen@uu.nl

Abstract

A disconnected cut of a connected graph is a vertex cut that itself also induces a disconnected subgraph. The corresponding decision problem is called Disconnected Cut. It is known that Disconnected Cut is NP-hard on general graphs, while polynomial-time algorithms exist for several graph classes. However, the complexity of the problem on claw-free graphs remained an open question. Its connection to the complexity of the problem to contract a claw-free graph to the 4-vertex cycle C_4 led Ito et al. (TCS 2011) to explicitly ask to resolve this open question. We prove that Disconnected Cut is polynomial-time solvable on claw-free graphs, answering the question of Ito et al. The basis for our result is a decomposition theorem for claw-free graphs of diameter 2, which we believe is of independent interest and builds on the research line initiated by Chudnovsky and Seymour (JCTB 2007–2012) and Hermelin et al. (ICALP 2011). On our way to exploit this decomposition theorem, we characterize how disconnected cuts interact with certain cobipartite subgraphs, and prove two further algorithmic results, namely that Disconnected Cut is polynomial-time solvable on circular-arc graphs and line graphs.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases disconnected cut, surjective homomorphism, biclique cover, claw-freeness

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.61

Related Version A full version is available at `https://arxiv.org/abs/1803.03663`.

Funding This paper received support from the Leverhulme Trust (RPG-2016-258).

1 Introduction

Graph connectivity is a crucial graph property studied in the context of network robustness. Well-studied notions of connectivity consider for example hamiltonicity, edge-disjoint spanning trees, edge cuts, vertex cuts, etc. In this paper, we study the notion of a *disconnected cut*, which is a vertex set U of a connected graph G such that $G - U$ is disconnected and the subgraph $G[U]$ induced by U is disconnected as well. Alternatively, we say that $V(G)$ can be partitioned into nonempty sets V_1, V_2, V_3, V_4 such that no vertex of V_1 is adjacent to a vertex of V_3 (that is, V_1 is anti-complete to V_3) and V_2 is anti-complete to V_4; then both $V_1 \cup V_3$ and $V_2 \cup V_4$ form a disconnected cut. See Figure 1 for an example. The Disconnected Cut problem asks whether a given connected graph G has a disconnected cut.

© Barnaby Martin, Daniël Paulusma, and Erik Jan van Leeuwen;
licensed under Creative Commons License CC-BY

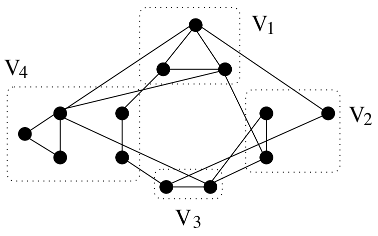


Figure 1 Graph with disconnected cuts $V_1 \cup V_3$ and $V_2 \cup V_4$ (figure originally appeared in [22]).

The Disconnected Cut problem is intimately connected to at least five other problems studied in the literature. We give a brief overview here, and refer to the related work section for more details. The name Disconnected Cut originates from Fleischner et al. [15], who determined the complexity of partitioning the vertices of a graph into exactly k *bicliques* (complete bipartite graphs with at least one edge), except for the case $k = 2$. For $k = 2$, this problem is polynomially equivalent to Disconnected Cut (by taking the complement of the input graph). The Disconnected Cut problem can also be seen as an H-Partition problem for appropriately defined 4-vertex graphs H. Dantas et al. [8] proved that H-Partition is polynomial-time solvable for each 4-vertex graph H except for the two cases equivalent to Disconnected Cut. If the input graph has diameter 2, then Disconnected Cut is equivalent to $\mathcal{C}_4$-Compaction [15], which asks for a homomorphism f from a graph G to the graph $\mathcal{C}_4$ (the 4-vertex-cycle with a self-loop in each vertex) such that for every $xy \in E(H)$ with $x \neq y$ there is an edge $uv \in E(G)$ with $f(u) = x$ and $f(v) = y$. The diameter-2 case is also equivalent to testing if a graph can be modified to a biclique by a series of edge contractions [22]. The restriction to graphs of diameter 2 is natural, as every graph of diameter 1 has no disconnected and every graph of diameter at least 3 has a disconnected cut [15]. Finally, Disconnected Cut fits in the broad study of vertex cut problems with extra properties on the cut set; see [23] for an overview.

The above demonstrates that Disconnected Cut is of central importance to understanding many different types of problems, ranging from cut problems to homomorphism and graph contractibility problems. Therefore, there has been broad interest to determine its computational complexity. Indeed, numerous papers [6, 8, 9, 10, 15, 21, 22, 28] asked about its complexity on general graphs. NP-completeness was proven independently in [26] and by Vikas, as announced in [30]. The strong interest in Disconnected Cut also led to a study on graph classes. We know polynomial-time algorithms for many classes [6, 9, 15, 22], particularly for certain classes of H-free graphs (graphs without a fixed graph H as an induced subgraph). However, even for several simple graphs H, the complexity landscape of Disconnected Cut still contains gaps. Indeed, even for four-vertex graphs H, we show (in the full version of our paper) that one open case remains, namely the case $H = K_4$. To prove this result, we need to deal with one non-trivial case, namely, when $H = K_{1,3}$ (the claw).

Our interest in Disconnected Cut on claw-free graphs is heightened by the close relation of this problem to C_r-Contractibility, which is to decide if a graph G contains the r-vertex cycle C_r as a contraction. This problem is NP-complete if $r \geq 4$ [3] and stays NP-complete for claw-free graphs as long as $r \geq 6$ [14]. The case $r \leq 3$ is polynomial-time solvable even for general graphs [3]. Hence, for claw-free graphs this leaves open the cases where $r \in \{4, 5\}$. Ito et al. [22] showed that C_4-Contractibility on claw-free graphs of diameter 2 is equivalent to Disconnected Cut. As Disconnected Cut is trivial if the input graph does not have diameter 2, this led Ito et al. [22] to explicitly ask the following:

What is the computational complexity of Disconnected Cut *on claw-free graphs?*

Our Contribution. We answer the open question of Ito et al. [22] by giving a polynomial-time algorithm for Disconnected Cut on claw-free graphs (which also finds a disconnected cut if it exists). This immediately implies that $\mathcal{C}_4$-Compaction and C_4-Contractibility are polynomial-solvable on claw-free graphs of diameter 2. As claw-free graphs are not closed under edge contraction, the latter is certainly not expected beforehand.

We start with the basic observation that Disconnected Cut is trivial if the graph does not have diameter 2 [15]. Hence, we aim for a deeper understanding of claw-free graphs of diameter 2. To this end, we give a new graph-theoretic theorem that proves that claw-free graphs of diameter 2 belong to one of four basic graph classes after performing two types of elementary operations. The theorem builds on one of the algorithmic decomposition theorems for claw-free graphs developed by Hermelin et al. [19, 20], and relies on the pioneering works of Chudnovsky and Seymour [5]. Several other algorithmic decomposition theorems for claw-free graphs have been built on the ideas of Chudnovsky and Seymour, see e.g. [11, 24], which jointly have had a broad impact on our algorithmic understanding of claw-free graphs (see [19] for an overview or [2]). Our structural theorem and resulting algorithm for Disconnected Cut expand this line of research.

The crux of the proof of our structural theorem is to exploit the extra structure offered by claw-free graphs of diameter 2 to show that the so-called strip-structures, which are central to the aforementioned decomposition theorems, only contain trivial strips. An important ingredient in the proof is to exclude not only twins (vertices u, v for which $N[u] = N[v]$), but also vertices with nested neighbourhoods (vertices u for which there exists a vertex v such that $N(u) \setminus \{v\} \subseteq N(v) \setminus \{u\}$). Using this operation, one can simplify the decomposition theorem of [19], and we think this observation may have an impact beyond this work.

Using the structural theorem, Disconnected Cut on claw-free graphs reduces to understanding its behavior under the elementary operations and on the basic graph classes. The crucial elementary operation is to remove certain cobipartite structures known as W-joins [5]. We develop the notion of unshatterable proper W-joins, which are essentially W-joins that cannot be broken into smaller W-joins, and exhibit how unshatterable proper W-joins interact with disconnected cuts. We then show that unshatterable proper W-joins can be removed from the graph by a simple operation. We complete our arguments by proving that all W-joins in the graph must be in fact be unshatterable proper W-joins, and that we can find unshatterable proper W-joins in polynomial time.

The main basic graph classes in the structural theorem are line graphs and proper circular-arc graphs. Prior to our work, the complexity of Disconnected Cut was unknown for these classes as well. We present a polynomial-time algorithm for line graphs and even for general circular-arcs graphs (not only proper circular-arcs). Both algorithms rely on the existence of a small induced cycle passing through a disconnected cut in a highly structured manner. In addition, for line graphs, we prove that the pre-image of the line graph is $2P_2$-free, and thus has diameter at most 3. The hardest part of the proof is then to prove that if the pre-image has diameter exactly 3, then the line graph has no disconnected cut.

Related Work. As mentioned, the name Disconnected Cut stems from Fleischner et al. [15], who studied how to partition the vertices of a graph into exactly k bicliques, where Disconnected Cut is equivalent to the case $k = 2$. However, Disconnected Cut originates from H-partitions, introduced in [8]. A model graph H on vertices $h_1, \ldots, h_k$ has solid and dotted edges. An H*-partition* of a graph G is a partition of $V(G)$ into $|V(H)|$ *nonempty* sets $V_1, \ldots, V_k$ such that for every pair of vertices $u \in V_i$ and $v \in V_j$: if h_ih_j is a solid edge of $E(H)$, then $uv \in E(G)$; and if h_ih_j is a dotted edge of $E(H)$, then $uv \notin E(G)$

(if $h_ih_j \notin E(H)$, then $uv \in E(G)$ or $uv \notin E(G)$ are both allowed). The corresponding decision problem is called H-Partition. Dantas et al. [8] proved that H-Partition is polynomial-time solvable for every 4-vertex model graph H except $H = 2K_2$, which has solid edges h_1h_3, h_2h_4 and no dotted edges, and $H = 2S_2$, which has dotted edges h_1h_3, h_2h_4 and no solid edges. These two cases are polynomial-time equivalent to Disconnected Cut. Hence, we now know that, as a matter of exception, H-Partition is NP-complete if $H \in \{2K_2, 2S_2\}$ [26].

We can encode a model graph H as a matrix M in which every entry is either 0 (dotted edge), 1 (solid edge) or $*$ (no restriction). If we allow sets V_i in a solution for H-Partition to be empty, then we obtain the M-Partition problem, introduced by Feder et al. [13]. This well-known problem generalizes many classical problems involving vertex cuts and partitions, including k-Colouring and H-Colouring; see also [18]. An even more general variant is to give every vertex u a list $L(u) \subseteq \{1, \ldots, k\}$ and to search for a solution, in which each vertex u may only belong to a set V_i with $i \in L(u)$. This yields the List M-Partition problem, which includes well-known cases, such as the Stubborn problem, which turned out to be polynomial-time solvable [7], in contrast to Disconnected Cut. A homomorphism f from G to H is a *retraction* if G contains H as an induced subgraph and $f(u) = u$ for every $u \in V(H)$. The corresponding decision version is called H-Retraction. Let $\mathcal{C}_4$ be the 4-cycle with a self-loop in each vertex. Then $\mathcal{C}_4$-Retraction is a special case of List $2S_2$-Partition where the input graph contains a cycle on four specified vertices $v_1, \ldots, v_4$ with $L(v_i) = \{i\}$ for $i = 1, \ldots, 4$ and $L(v) = \{1, 2, 3, 4\}$ for $v \notin \{v_1, \ldots, v_4\}$. This problem is a generalization of Disconnected Cut. Feder and Hell [12] proved that $\mathcal{C}_4$-Retraction is NP-complete. Hence, List $2S_2$-Partition and List $2K_2$-Partition are NP-complete. Note that this result is also implied by the NP-completeness of $2K_2$-Partition [26].

Vikas [29] solved an open problem of Winkler (see [13, 29]) by proving NP-completeness of $\mathcal{C}_4$-Compaction, the variant of the $2S_2$-Partition problem with the extra constraint that there must be at least one edge u_iu_j with $u_i \in V_i$ and $u_{i+1} \in V_{i+1}$ for $i = 1, \ldots, 4$ (where $V_5 = V_1$). Generally, a homomorphism f from a graph G to a graph H is a *compaction* if f is edge-surjective, i.e., for every $xy \in E(H)$ with $x \neq y$ there is an edge $uv \in E(G)$ with $f(u) = x$ and $f(v) = y$. The corresponding decision problem is called H-Compaction. If $H = \mathcal{C}_4$, then the problem is equivalent to Disconnected Cut when restricted to graphs of diameter 2 [15]. Hence, $\mathcal{C}_4$-Compaction is NP-complete for graphs of diameter 2 [26] (the result of [29] holds for graphs of diameter at least 3). Similarly, a homomorphism f from a graph G to a graph H is *(vertex-)surjective* if for every $x \in V(H)$ there is a vertex $u \in V(G)$ such that $f(u) = x$. The decision problem is called Surjective H-Colouring (or H-Vertex Compaction, or Surjective H-Homomorphism) and is equivalent to Disconnected Cut if $H = \mathcal{C}_4$. The complexity classifications of H-Compaction and Surjective H-Colouring are wide open despite many partial results; see [1] for a survey and [16] for a more recent overview focussing on Surjective H-Colouring .

2 Preliminaries and Basic Results

In the remainder of our paper, graphs are finite, undirected, and have neither multiple edges nor self-loops unless explicitly stated otherwise. Let $G = (V, E)$ be a graph. For a set $S \subseteq V$, $G[S]$ is the subgraph of G induced by S. We say that S is *connected* if $G[S]$ is connected. We write $G - S = G[V \setminus S]$, and if $S = \{u\}$, we write $G - u$ instead. For a vertex $u \in V$, let $N(u) = \{v \mid uv \in E\}$ be the neighbourhood of u and $N[u] = N(u) \cup \{u\}$. The *complement*

$\overline{G}$ of G has vertex set V and edge set $\{uv \mid uv \notin E\}$. The *distance* $d_G(u,v)$ between vertices u and v of G is the number of edges in a shortest path between them. The *diameter* of G is equal to $\max\{d_G(u,v) \mid u,v \in V\}$. The following lemma was observed by Fleischner et al.

▶ **Lemma 1** ([15]). *If a graph G has diameter 1, then G has no disconnected cut. If a graph G has diameter at least 3, then G has a disconnected cut, which can be found in linear time.*

A subset $D \subseteq V$ is a *dominating* set of a graph $G = (V,E)$ if every vertex of $V \setminus D$ is adjacent to at least one vertex of D. If $D = \{u\}$, then u is a *dominating* vertex of G. A vertex $u \in V$ has a *disconnected neighbourhood* if $N(u)$ induces a disconnected graph.

▶ **Lemma 2.** *If a graph G contains a dominating vertex, then G has no disconnected cut.*

▶ **Lemma 3** (proof omitted). *If a graph G contains a non-dominating vertex u with a disconnected neighbourhood, then G has a disconnected cut.*

Two disjoint vertex sets S and T in a graph $G = (V,E)$ are *complete* to each other if there is an edge between every vertex of S and every vertex of T, and S and T are *anticomplete* to each other if there is no edge between a vertex of S and a vertex of T. Recall that G has a *disconnected cut* if V can be partitioned into four nonempty sets V_1, V_2, V_3, V_4, such that V_1 is anticomplete to V_3 and V_2 is anticomplete to V_4. We say that V_1, V_2, V_3, V_4 form a *disconnected partition* of G.

▶ **Lemma 4.** *Let V_1, V_2, V_3, V_4 be a disconnected partition of a graph G of diameter 2. Then G has an induced cycle C with $4 \le |V(C)| \le 5$ such that $V(C) \cap V_i \neq \emptyset$ for $i = 1, \ldots, 4$.*

Proof. Let $u_1 \in V_1$ and $u_3 \in V_3$. As G has diameter 2, there exists a vertex u_2 in V_2 or V_4, say V_2, such that u_2 is adjacent to u_1 and to u_3. Let $u_4 \in V_4$. As G has diameter 2, there exists a vertex u_1' in V_1 or V_3, say V_1, such that u_1' is adjacent to u_2 and u_4. If u_3 and u_4 are adjacent, then we can take as C the cycle on vertices u_1', u_2, u_3, u_4 in that order. Otherwise, as G has diameter 2, there exists a vertex $w \in V_3 \cup V_4$, such that w is adjacent to u_3 and to u_4. In that case we can take as C the cycle on vertices u_1', u_2, u_3, w, u_4. ◀

Two adjacent vertices u and v of graph $G = (V,E)$ have a *nested neighbourhood* if $N(u)\setminus\{v\} \subseteq N(v)\setminus\{u\}$ or $N(v)\setminus\{u\} \subseteq N(u)\setminus\{v\}$. We say that G has *distinct neighbourhoods* if G has no two vertices that have nested neighbourhoods.

▶ **Lemma 5** (proof omitted). *Let G be a graph of diameter 2 that contains two vertices u and v such that $N(u) \setminus \{v\} \subseteq N(v) \setminus \{u\}$. Then G has a disconnected cut if and only if $G - u$ has a disconnected cut. Moreover, $G - u$ has diameter at most 2.*

A pair of vertices u and v of a graph $G = (V,E)$ is a *universal pair* if $\{u,v\}$ is a dominating set and there exist distinct vertices x and y in $V \setminus \{u,v\}$, such that $x \in N(u)$ and $y \in N(v)$; note that this implies that $|V| \ge 4$ and u,v have at least one neighbour in $V - \{u,v\}$. Let H be a graph. Then G is *H-free* if G contains no induced subgraph isomorphic to H. The *disjoint union* $G + H$ of two vertex-disjoint graphs G and H is the graph $(V(G) \cup V(H), E(G) \cup E(H))$. The disjoint union of r copies of a graph G is denoted by rG. The graphs C_r and P_r denote the cycle and path on r vertices, respectively. The graph K_r denotes the complete graph on r vertices. The *independence number* $\alpha(G)$ of a graph G is the largest k such that G contains an induced subgraph isomorphic to kP_1.

▶ **Lemma 6** ([6]). *A $2P_2$-free graph has a disconnected cut if and only if its complement has a universal pair.*

▶ **Lemma 7** ([9]). Disconnected Cut *is $O(n^3)$-time solvable for $4P_1$-free graphs.*

The graph $(\{u, v_1, v_2, v_2\}, \{uv_1, uv_2, uv_3\})$ is the *claw* $K_{1,3}$. A graph is *cobipartite* if it is the complement of a bipartite graph. The *line graph* of a graph G with edges $e_1, \ldots, e_p$ is the graph $L(G)$ with vertices $u_1, \ldots, u_p$ such that there is an edge between any two vertices u_i and u_j if and only if e_i and e_j have a common endpoint in G. Note that every line graph is claw-free. We call G the *preimage* of $L(G)$. Every connected line graph except K_3 has a unique preimage [17]. A *circular-arc graph* is a graph that has a representation in which each vertex corresponds to an arc of a circle, such that two vertices are adjacent if and only if their corresponding arcs intersect. An *interval graph* is a graph that has representation in which each vertex corresponds to an interval of the line, such that two vertices are adjacent if and only if their corresponding intervals intersect. Note that circular-arc graphs generalize interval graphs. A circular-arc or interval graph is *proper* if it has a representation where the arcs respectively intervals are such that no one is contained in another.

3 Circular-Arc Graphs

In this section we prove that Disconnected Cut is polynomial-time solvable for circular-arc graphs. This result is known already for interval graphs, as it follows from the result that Disconnected Cut is polynomial-time solvable for the class of chordal graphs [22], which contains the class of interval graphs. In fact, we have an $O(n^2)$-time algorithm for interval graphs. Due to Lemma 4 and the fact that interval graphs are chordal, no interval graph of diameter 2 has a disconnected cut. Consequently, an interval graph has a disconnected cut if and only if its diameter is at least 3 due to Lemma 1. To show that Disconnected Cut is polynomial-time solvable for circular-arc graphs requires significant additional work.

Let G be a circular-arc graph. For each vertex $u \in V(G)$ we can associate an arc $[l_u, r_u]$ where we say that l_u is the clockwise left endpoint of u and r_u is the clockwise right endpoint of u. We may assume that all left and right endpoints of the vertices of G are unique.

▶ **Lemma 8** ([27]). *A circular-arc graph G on n vertices and m edges can be recognized in $O(n + m)$ time. In the same time, a representation of G can be constructed with distinct arc endpoints that are clockwise enumerated as $1, \ldots, 2n$.*

For the main result in the section we need the following lemma (proof omitted).

▶ **Lemma 9.** *Let G be a circular-arc graph of diameter 2 with a disconnected cut. Then G has a disconnected partition V_1, V_2, V_3, V_4 such that each V_i is connected.*

▶ **Theorem 10.** Disconnected Cut *is $O(n^2)$-time solvable for circular-arc graphs.*

Proof Sketch. Let $G = (V, E)$ be a circular-arc graph on n vertices. We will either find a disconnected cut or conclude that G has no disconnected cut. We compute the diameter of G in $O(n^2)$ time, say by using the (more general) $O(n^2)$-time algorithm of [4]. By Lemma 1, we may assume that G has diameter 2. Lemma 9 tells us that if G has a disconnected cut, then G has a disconnected partition V_1, V_2, V_3, V_4 such that V_i is connected for $i = 1, \ldots, 4$. We say that the *arc of a set* V_i is the union of all the arcs of the vertices in V_i. As G has diameter 2, the union of the arcs of the sets V_i covers the whole circle. Moreover, the arcs of V_1 and V_3 are disjoint and the arcs of V_2 and V_4 are disjoint.

We now compute, in linear time, a representation of G with distinct arc endpoints clockwise enumerated as $1, \ldots, 2n$ via Lemma 8. We then sort the arcs in $O(n \log n)$ time. Then, in $O(n^2)$ time, we check if there is a pair or triple of vertices whose arcs cover the whole circle. If so, then using Lemma 9 we find that G has no disconnected cut.

Suppose G has no pair or triple of vertices whose arcs cover the whole circle. Then, in $O(n)$ time, we find an induced cycle $v_1, \ldots, v_k$ whose arcs cover the whole circle and such that $k \in \{4, 5\}$. If G has a disconnected partition V_1, V_2, V_3, V_4 such that each V_i is connected, then we have the following. If $k = 4$, then we may assume without loss of generality that $v_i \in V_i$ for $i = 1, \ldots, 4$. If $k = 5$, then two vertices v_i, v_{i+1} belong to the same set V_h, whereas sets V_i with $i \neq h$ each contain a single vertex from C. If $k = 5$, then we guess which two vertices v_i, v_{i+1} will be put in the same set, say v_1, v_5; this does not influence the asymptotic running time.

Now we build up the sets V_i from scratch by putting in the vertices from $V(G) \setminus \{v_1, \ldots, v_k\}$. We maintain that each V_i induces a connected graph, and thus, the union of the arcs of the vertices in V_i indeed always form an arc. Observe also that no set V_h is contained in some other set V_i; by our choice of vertices v_i we will always maintain this property. We say that a vertex u *intersects* a set V_i if the arc of u intersects the arc of V_i. Note that, since the arcs corresponding to $\{v_1, \ldots, v_k\}$ cover the entire circle, so do the arcs of the sets V_i that we are constructing. If there is a vertex that intersects each of the sets V_i constructed so far, then there is no disconnected cut with each V_i connected. If $k = 4$, then G has no disconnected cut due to Lemma 9. If $k = 5$, then our guess of vertices v_1, v_5 to belong to V_h may have been incorrect, and we need to put two other consecutive vertices of C in the same set V_h before concluding that G has no disconnected cut. Otherwise, we do the following until no longer possible. As no V_h is contained in some other V_h, any vertex u that intersects two sets V_i and V_{i+2} for some i (say, $i \in \{1, 2\}$ without loss of generality), also intersects V_{i+1} or V_{i+3} (where $V_5 = V_1$). We put a vertex u that intersects two sets V_i and V_{i+2} for some i into set V_{i+1} if u intersects V_{i+1} as well; otherwise, u intersects V_{i+3} and we put u in V_{i+3}.

Let T be the set of vertices of G that we have not placed in some set V_i yet. We show that each vertex of T must intersect exactly two sets V_i and V_j such that, in addition, $j = i + 1$ holds. Then we can model the remaining instance as an instance of 2-SATISFIABILITY and solve it in $O(n^2)$ time. ◀

4 Line Graphs

In this section we prove that DISCONNECTED CUT is polynomial-time solvable for line graphs. We start with the following lemma due to Ito et al. [22].

▶ **Lemma 11** ([22]). *Let G be a graph with diameter 2 whose line graph $L(G)$ also has diameter 2. Then G has a disconnected cut if and only if $L(G)$ has a disconnected cut.*

For the main result in the section we need the following lemma (proof omitted).

▶ **Lemma 12.** *Let G be a graph that is neither a triangle nor a star. Then $L(G)$ has diameter 2 if and only if G is $2P_2$-free.*

▶ **Theorem 13.** DISCONNECTED CUT *is $O(n^4)$-time solvable on line graphs of n-vertex graphs.*

Proof. Let G be a graph on n vertices and m edges. We first check in $O(n)$ time if G is a triangle or star. If so, then $L(G)$ is a complete graph and thus $L(G)$ has no disconnected cut. From now on suppose that G is neither a triangle nor a star. By Lemma 12 we find that $L(G)$ has diameter 2 if and only if G is $2P_2$-free. Hence, we can check in $O(n^4)$ time, via checking if G has an induced $2P_2$ by brute force, if $L(G)$ has diameter 2.

First assume that $L(G)$ does not have diameter 2. As G is not a triangle or a star, $L(G)$ has diameter at least 3. By Lemma 1 we find that $L(G)$ has a disconnected cut. Now assume that $L(G)$ has diameter 2. We check in $O(n^3)$ time if G has an edge uv such that every vertex of $V(G) \setminus \{u,v\}$ is adjacent to at least one of u,v. If so, then uv is a dominating vertex of $L(G)$, and $L(G)$ has no disconnected cut due to Lemma 2. If not, then $L(G)$ has no dominating vertices, and we proceed as follows. First we check if $L(G)$ has a vertex uv with a disconnected neighbourhood, or equivalently, if G contains an edge uv such that u and v have degree at least 2 and no common neighbours. This takes $O(n^3)$ time. If $L(G)$ has a vertex with a disconnected neighbourhood, then $L(G)$ has a disconnected cut by Lemma 3. From now on assume that $L(G)$ has no vertex with a disconnected neighbourhood. As G is neither a triangle nor a star, G is $2P_2$-free by Lemma 12. Hence, G has diameter at most 3. We can determine in $O(n^3)$ time the diameter of G and consider each case separately.

Case 1. G has diameter 1.
We claim that $L(G)$ has no disconnected cut. For contradiction, assume that $L(G)$ has a disconnected cut. Let V_1', V_2', V_3', V_4' be a disconnected partition of $L(G)$. By Lemma 4, $L(G)$ contains a cycle C' with vertices u_iu_{i+1} for $i = 1,\ldots,j$ (with $u_{j+1} = u_1$) and $j \in \{4,5\}$, such that $V(C') \cap V_i' \neq \emptyset$ for $i = 1,2,3,4$. Then we may assume without loss of generality that $u_iu_{i+1} \in V_i'$ for $i = 1,\ldots,4$ and $u_ju_{j+1} \in V_4'$. As G has diameter 1, u_1u_3 is an edge of G and thus a vertex of $L(G)$. In $L(G)$, u_1u_3 is adjacent to every vertex in $\{u_1u_2, u_2u_3, u_3u_4, u_ju_{j+1}\}$, and thus to a vertex in V_i' for $i = 1,\ldots,4$, a contradiction.

Case 2. G has diameter 2.
Then G has a disconnected cut if and only if $L(G)$ has a disconnected cut due to Lemma 11. By Lemma 6 it suffices to check if $\overline{G}$ has a universal pair. This takes $O(n^3)$ time.

Case 3. G has diameter 3.
We will prove that $L(G)$ has no disconnected cut. As G has diameter 3, G does have a disconnected cut by Lemma 1. We need the following claim.

▶ **Claim.** *Let V_1, V_2, V_3, V_4 be a disconnected partition of G. Then every cycle C of G with $4 \leq |V(C)| \leq 5$ contains vertices of at most three distinct sets from $\{V_1, V_2, V_3, V_4\}$.*

We prove the Claim as follows. For contradiction, assume that G has a cycle C with vertices $u_1,\ldots,u_j$ for $j \in \{4,5\}$, such that $V(C) \cap V_i \neq \emptyset$ for $i = 1,\ldots,4$. We may assume without loss of generality that $u_i \in V_i$ for $i = 1,\ldots,4$ and $u_j \in V_4$. As G is $2P_2$-free, we may assume without loss of generality that u_3 is in a singleton connected component of $G[V_3]$. If $j = 4$, then we may also assume without loss of generality that u_2 is in a singleton connected component of $G[V_2]$. If $j = 5$, then u_2 must be in a singleton connected component of $G[V_2]$ due to the edge u_4u_5, which is contained in $G[V_4]$. This means that the sets $N_G(u_2) \setminus \{u_3\}$ and $N_G(u_3) \setminus \{u_2\}$ are disjoint. As u_1u_2 and u_3u_4 are edges of G, both $N_G(u_2) \setminus \{u_3\}\}$ and $N_G(u_3) \setminus \{u_2\}\}$ are nonempty. Hence, the vertex u_2u_3 has a disconnected neighbourhood in $L(G)$, a contradiction. This proves the Claim.

Now, for contradiction, assume that $L(G)$ has a disconnected cut. Let V_1', V_2', V_3', V_4' be a disconnected partition of $L(G)$. By Lemma 4, $L(G)$ contains a cycle C' with vertices u_iu_{i+1} for $i = 1,\ldots,j$ (with $u_{j+1} = u_1$) and $j \in \{4,5\}$, such that $V(C') \cap V_i' \neq \emptyset$ for $i = 1,2,3,4$. Assume without loss of generality that $u_iu_{i+1} \in V_i'$ for $i = 1,\ldots,4$ and $u_ju_{j+1} \in V_4'$.

We define the following partition V_1, V_2, V_3, V_4 of $V(G)$. Let $u \in V(G)$. If u is incident to only edges from one set V_i', then we put u in V_i. Suppose u is incident to edges from more than one set V_i'. As V_1', V_2', V_3', V_4' is a disconnected partition of $L(G)$, we find that u is incident to edges from V_i' and V_{i+1}' for some $1 \leq i \leq 4$ (where $V_5 = V_1$) and to no other sets V_j'. In that case we put u into V_{i+1}.

We now prove that V_1 is anticomplete to V_3. For contradiction, assume that V_1 contains a vertex u and V_3 contains a vertex v such that $uv \in E(G)$. As $u \in V_1$, we find that u is incident to edges only in V_4' and V_1'. Hence, $uv \in V_1' \cup V_4'$. As $v \in V_3$, we find that v is incident to edges only in V_2' and V_3'. This implies that $uv \in V_2' \cup V_3'$, a contradiction. By the same argument we can show that V_2 is anticomplete to V_4. Let C be the cycle with vertices $u_1, \ldots, u_j$ in G. Then $V(C) \cap V_i \neq \emptyset$, and thus $V_i \neq \emptyset$, for $i = 1, \ldots, 4$. Hence, V_1, V_2, V_3, V_4 is a disconnected partition of G, and C is a cycle in G with $V(C) \cap V_i \neq \emptyset$ for every i. This is not possible due to the Claim. We conclude that $L(G)$ has no disconnected cut.

The correctness of our algorithm follows from the above. If G has diameter 1 (Case 1) or diameter 3 (Case 3), no additional running time is required, as $L(G)$ has no disconnected cut in both these cases. Hence, only executing Case 2 takes additional time, namely time $O(n^3)$. Hence, the total running time of our algorithm is $O(n^4)$. ◀

5 Claw-Free Graphs

In this section, we prove that Disconnected Cut is polynomial-time solvable on claw-free graphs. The proof consists of two parts. In Section 5.1 we show how to get rid of certain cobipartite structures in the graph, called W-joins. We remark that Disconnected Cut can be solved in polynomial time on cobipartite graphs [15]. Although this is a necessary condition for Disconnected Cut to be solvable in polynomial time on claw-free graphs, the algorithm for cobipartite graphs is not sufficient to deal with W-joins. In Section 5.2 we present our new decomposition theorem for claw-free graphs of diameter 2 and combine this theorem with the results from the previous sections and Section 5.1 to show our main result.

5.1 Cobipartite Structures versus Disconnected Cuts

A pair (A, B) of disjoint non-empty sets of vertices is a *W-join* in graph G if $|A| + |B| > 2$, A and B are cliques, A is neither complete nor anticomplete to B, and every vertex of $V(G) \setminus (A \cup B)$ is either complete or anticomplete to A and either complete or anticomplete to B. A W-join is a *proper W-join* if each vertex in A is neither complete nor anticomplete to B and each vertex in B is neither complete nor anticomplete to A. Observe that for a proper W-join (A, B), it must hold that $|A|, |B| \geq 2$. For any W-join (A, B), it holds that $G[A \cup B]$ is a cobipartite induced subgraph in G.

We assume that an input graph G of Disconnected Cut has diameter 2 and that G has distinct neighbourhoods, by Lemmas 1 and 5 respectively. We show how to use these assumptions to remove all W-joins in a claw-free graph and obtain an equivalent instance of Disconnected Cut. As a first step, we show that we can focus on proper W-joins.

▶ **Lemma 14** (proof omitted). *Let G be a graph with distinct neighbourhoods. If G admits a W-join (A, B), then (A, B) is a proper W-join.*

A W-join (A, B) is *partitionable* if there are partitions of A into non-empty sets A', A'' and of B into non-empty sets B', B'' such that A' is anticomplete to B'' and B' is anticomplete to A''. A proper W-join (A, B) is *shatterable* if it is partitionable with sets A', A'', B', B'' and one of $(A', B'), (A'', B'')$ is also a proper W-join; we say it is *unshatterable* otherwise.

▶ **Lemma 15** (proof omitted). *Let G be a graph with distinct neighbourhoods and let (A, B) be a proper W-join in G. If (A, B) is partitionable and unshatterable, then $G[A \cup B]$ is isomorphic to C_4.*

▶ **Lemma 16** (proof omitted)**.** *Let G be a claw-free graph that is not cobipartite, has distinct neighbourhoods, and has diameter 2. Let (A,B) be a proper W-join in G that is unshatterable. If G admits a disconnected cut, then there exists a disconnected partition V_1,V_2,V_3,V_4 of G such that $V_i \cap (A \cup B) = \emptyset$ for some $i \in \{1,2,3,4\}$.*

Let (A,B) be a proper W-join of a graph G. For any two adjacent vertices $a \in A$ and $b \in B$, let G_{ab} be the graph obtained from G by removing $A \setminus \{a\}$ and $B \setminus \{b\}$. Observe that the graph G_{ab} is the same regardless of the choice of a, b.

▶ **Lemma 17.** *Let G be a claw-free graph that is not cobipartite, has distinct neighbourhoods, and has diameter 2. Let (A,B) be a proper W-join of G that is unshatterable. Then G admits a disconnected cut if and only if G_{ab} admits a disconnected cut for any two adjacent vertices $a \in A$ and $b \in B$.*

Proof. First suppose that G_{ab} admits a disconnected partition V_1,V_2,V_3,V_4 for any two vertices a,b. Let $a \in V_i$ and $b \in V_j$ for $i,j \in \{1,2,3,4\}$. Then the sets V_1',V_2',V_3',V_4' obtained from V_1,V_2,V_3,V_4 by adding A to V_i and B to V_j is a disconnected partition of G.

Now suppose that G admits a disconnected cut. Let V_1,V_2,V_3,V_4 be a disconnected partition of G. By Lemma 16, we may assume without loss of generality that $V_4 \cap (A \cup B) = \emptyset$. Note that A is a clique in G and V_1 is anticomplete to V_3, and thus $A \subseteq V_1 \cup V_2$ or $A \subseteq V_2 \cup V_3$. We assume the former without loss of generality. Among all such disconnected partitions, we will assume that V_1,V_2,V_3,V_4 was chosen to minimize $|A \cap V_1|$.

We consider several cases. In each of these cases we find two vertices a,b for which we can construct a disconnected partition of G_{ab}. Note that this suffices to prove the statement, as the graph G_{ab} is the same regardless of the choice of a,b.

First assume that $A \subseteq V_1$. Since no vertex of B is anticomplete to A by the definition of a proper W-join and V_1 is anticomplete to V_3, it follows that $B \subseteq V_1 \cup V_2$. Now if $B \subseteq V_1$, then let $a \in A$ and $b \in B$ be arbitrary adjacent vertices (these exist by the definition of a W-join) and $V_1 \setminus ((A \setminus \{a\}) \cup (B \setminus \{b\})), V_2, V_3, V_4$ is a disconnected partition of G_{ab}. Otherwise, let $b \in B \cap V_2$ and let a be an arbitrary vertex of A that is adjacent to b (which exists by the definition of a proper W-join). Then $V_1 \setminus ((A \setminus \{a\}) \cup B), V_2 \setminus (B \setminus \{b\}), V_3, V_4$ is a disconnected partition of G_{ab}.

Now assume that $A \subseteq V_2$. Note that $B \subseteq V_1 \cup V_2 \cup V_3$. Since B is a clique and V_1 is anticomplete to V_3, it follows that $B \subseteq V_1 \cup V_2$ or $B \subseteq V_2 \cup V_3$. First, assume that $B \subseteq V_2$. Let $a \in A$ and $b \in B$ be arbitrary adjacent vertices; note that $a,b \in V_2$. Then $V_1, V_2 \setminus ((A \setminus \{a\}) \cup (B \setminus \{b\})), V_3, V_4$ is a disconnected partition of G_{ab}. So we may assume that $B \not\subseteq V_2$. Then $B \cap V_1 \neq \emptyset$ or $B \cap V_3 \neq \emptyset$. Without loss of generality, we assume it is the former. Let $b \in B \cap V_1$ and let $a \in A$ be any neighbour of b. Note that $a \in V_2$. Then $V_1 \setminus (B \setminus \{b\}), V_2 \setminus ((A \setminus \{a\}) \cup B), V_3, V_4$ is a disconnected partition of G_{ab}.

It remains to consider the case where $A \cap V_1 \neq \emptyset$ and $A \cap V_2 \neq \emptyset$. Let $P = N(A) \setminus N[B]$, $Q = N(B) \setminus N[A]$, $M = N[A \cup B] \setminus (P \cup Q)$, and $R = V(G) \setminus (P \cup Q \cup M)$. Note that P is complete to A and anticomplete to B, whereas Q is complete to B and anticomplete to A. Moreover, M is complete to $A \cup B$, whereas R is anticomplete to $A \cup B$. Then, by the assumptions of the case, we have that $P \subseteq V_1 \cup V_2$. Note that $B \subseteq V_1 \cup V_2 \cup V_3$. Since B is a clique and V_1 is anticomplete to V_3, it follows that $B \subseteq V_1 \cup V_2$ or $B \subseteq V_2 \cup V_3$. Moreover, as $A \cap V_1 \neq \emptyset$, it follows from the definition of a proper W-join that $B \not\subseteq V_3$. We now prove that $B \subseteq V_1 \cup V_2$.

For contradiction, assume that $B \cap V_3 \neq \emptyset$ and thus $B \cap V_2 \neq \emptyset$. As M is complete to A and B and $A \cup B$ has a nonempty intersection with each of V_1,V_2,V_3, it follows from the definition of a disconnected partition that $M \subseteq V_2$. Similarly, we derive that

$Q \subseteq V_2 \cup V_3$; recall also that $P \subseteq V_1 \cup V_2$. Suppose $V_1 \setminus A \neq \emptyset$. Then $V_1 \setminus A, V_2 \cup A, V_3, V_4$ is also a disconnected partition of G, contradicting our choice of the disconnected partition V_1, V_2, V_3, V_4. Hence, $V_1 \setminus A = \emptyset$ and thus, $V_1 \subseteq A$. Then $P \subseteq V_2$. By the definition of a W-join, any path of length 2 from a vertex in R to a vertex in A must intersect P or M. As $M \cup P \subseteq V_2$ and V_4 is anticomplete to V_2, we obtain $R \cap V_4 = \emptyset$. Since $A \cup B \cup P \cup M \cup Q \cup R = V(G)$ and none of A, B, P, M, Q, R intersects V_4, it follows that $V_4 = \emptyset$, a contradiction.

We may thus assume that $B \subseteq V_1 \cup V_2$. First, assume that there exist adjacent vertices $a \in A$ and $b \in B$ such that $|V_1 \cap \{a, b\}| = 1$ (and thus $|V_2 \cap \{a, b\}| = 1$). Then $V_1 \setminus ((A \cup B) \setminus \{a, b\}), V_2 \setminus ((A \cup B) \setminus \{a, b\})$ is a disconnected partition of G_{ab}. Hence, we may assume that no such two vertices exist. It follows that neither $B \subseteq V_1$ nor $B \subseteq V_2$; otherwise, such a and b would exist by the definition of a proper W-join. Then we may conclude that (A, B) is partitionable with sets $A \cap V_1, A \cap V_2, B \cap V_1, B \cap V_2$. Since (A, B) is unshatterable, it follows from Lemma 15 that $G[A \cup B]$ is isomorphic to C_4. Hence, $|A| = |B| = 2$ and V_1, V_2 each contain exactly one vertex of A and exactly one vertex of B. Since G is not cobipartite and G is connected (as G has diameter 2), it follows that one of P, M, Q is non-empty. However, each vertex in $P \cup M \cup Q$ is adjacent to a vertex of V_1 and a vertex of V_2. Hence, $P \cup M \cup Q \subseteq V_1 \cup V_2$. Without loss of generality, $(P \cup M \cup Q) \cap V_1 \neq \emptyset$. Let a be the single vertex of $A \cap V_2$ and let b be the single vertex of $B \cap V_2$. Then $V_1 \setminus (A \cup B), V_2, V_3, V_4$ is a disconnected partition of G_{ab}. The lemma follows. ◀

In Section 5.2 we will show that by iterating the above lemma, we can remove all W-joins from an input claw-free graph of diameter 2. However, to this end, it is crucial to have a polynomial-time algorithm that actually finds an unshatterable proper W-join (if it exists). Our algorithm for this problem relies on the $O(n^2m)$-time algorithm by King and Reed [25] to find a proper W-join (A, B). We test in linear time whether the proper W-join is partitionable by considering the graph H obtained from $G[A \cup B]$ by removing all edges with both endpoints in A or in B. We argue that we can recurse on a smaller proper W-join if H has two or more connected components, and that (A, B) is unshatterable otherwise.

▶ **Lemma 18** (proof omitted). *Let G be a graph with distinct neighbourhoods. Then in $O(n^2m)$ time, we can find an unshatterable proper W-join in G, or report that G has no proper W-join.*

5.2 Structure of Claw-Free Graphs and Solving Disconnected Cut

Before our main result we first show a decomposition of claw-free graphs of diameter 2.

▶ **Theorem 19.** *Every claw-free graph G of diameter 2 with distinct neighbourhoods, no W-joins, $\alpha(G) > 3$, and $|V(G)| > 13$ is a proper circular-arc graph or a line graph.*

Proof Sketch. One of the algorithmic structure theorems for claw-free graphs by Hermelin et al. [19, Theorem 6.8] (see also [20]) essentially shows that a claw-free graph that satisfies the assumptions of the theorem is almost a line graph, but certain vertices of this line graph are replaced with large structures called stripes. A stripe is basically an induced subgraph of the graph with one or two specially marked 'ends'. These ends are cliques, contain the only vertices that are incident with edges that connect the stripe with the rest of the graph, and for each end its neighbourhood outside the stripe is a clique. Using the fact that the diameter is 2, we argue that if a stripe contains a vertex x that is not in an end of the stripe, then every vertex of G must be in the stripe or in the neighbourhood of its ends. This enables us to prove the main claim in the theorem, which is that the stripe is essentially the whole graph if it has such a vertex x. After proving the claim, it suffices to consider the different cases in the structural theorem and prove that, by using the claim and the assumptions of the theorem, G must be a proper circular-arc graph or a line graph. ◀

▶ **Theorem 20.** Disconnected Cut *is* $O(n^3m)$*-time solvable for claw-free graphs.*

Proof. Let G be a connected claw-free graph on n vertices and m edges. We will either find a disconnected cut or conclude that G has no disconnected cut. Assume $n \geq 14$. We compute the diameter of G in $O(n^2)$ time. By Lemma 1, G has no disconnected cut if its diameter is 1 and has a disconnected cut if its diameter is at least 3. Assume the diameter of G is 2. We check if $\alpha(G) \leq 3$ in $O(n(m + n \log n))$ time [11]. If so, then we decide if G has a disconnected cut in $O(n^3)$ time by Lemma 7. Assume $\alpha(G) > 3$. Hence, G is not cobipartite.

Next, we check whether G contains a vertex u for which there exists a vertex v such that $N(u) \setminus \{v\} \subseteq N(v) \setminus \{u\}$. This takes $O(n^3)$ time. If so, then we remove u from G (and restart the algorithm with the resulting graph, which is still connected and claw-free). This is correct by Lemma 5. Hence, we may assume that G has distinct neighbourhoods.

Then, we get rid of all W-joins in G. Since G has distinct neighbourhoods, it follows from Lemma 14 that every W-join in G is a proper W-join. Using Lemma 18, in $O(n^2m)$ time, we can find an unshatterable W-join in G or correctly decide that G does not admit a proper W-join (and thus no W-join). In the former case, we apply Lemma 17 to the unshatterable proper W-join (A, B) that is found. This takes linear time. We then restart the algorithm on the graph G_{ab} found by Lemma 17 (note that G_{ab} is still connected and claw-free). Since $|A| + |B| \geq 3$, $|V(G_{ab})| < |V(G)|$ and thus we can recurse at most n times. Hence, we may assume that G admits no W-joins.

Next, we check if G is a circular-arc graph in linear time by Lemma 8. If so, then we apply Theorem 10 to decide if G has a disconnected cut in $O(n^2)$ time. Hence, we may assume that G is not (proper) circular-arc. By Theorem 19 this means that G is a line graph. Hence, we apply Theorem 13 to decide whether G admits a disconnected cut in $O(n^4)$ time. This finishes the description of the algorithm. The running time is clearly $O(n^3m)$. ◀

Recall from [15, 22] that C_4-Contractibility and $\mathcal{C}_4$-Compaction are equivalent to Disconnected Cut on graphs of diameter 2. We combine these claims with Theorem 19.

▶ **Corollary 21.** C_4-Contractibility *and* $\mathcal{C}_4$-Compaction *are* $O(n^3m)$*-time solvable for claw-free graphs of diameter* 2.

6 Open Problems

In light of Corollary 21 we ask about the complexities of C_4-Contractibility and $\mathcal{C}_4$-Compaction for claw-free graphs of diameter at least 3. We note that the NP-complete problem P_4-Contractibility [3] is polynomial-time solvable for claw-free graphs [14].

It is not known if there exists a graph H for which H-Compaction and Surjective H-Colouring have a different complexity. If we impose restrictions on the input graph, such a graph H is known: $\mathcal{C}_4$-Compaction is NP-complete for graphs of diameter 3 [29], whereas Surjective $\mathcal{C}_4$-Colouring (being equivalent to Disconnected Cut) is trivial on this graph class. In contrast to claw-free graphs, graphs of diameter 3 do not form a hereditary graph class, that is, they are not closed under vertex deletion. This leads to the natural question if there exist a hereditary graph class $\mathcal{G}$ and a graph H, such that H-Compaction and Surjective H-Colouring have different complexity when restricted to $\mathcal{G}$. Should

$\mathcal{C}_4$-Compaction turn out to be NP-complete for claw-free graphs, then due Theorem 19 and the equivalency between Disconnected Cut and Surjective H-Colouring we can take the class of claw-free graphs as $\mathcal{G}$ and the graph $\mathcal{C}_4$ as H to find such a pair $(\mathcal{G}, H)$.

We also ask what the complexity of Disconnected Cut is for K_4-free graphs; as shown in the full version of our paper, the K_4 is the only 4-vertex graph H for which this is still open.

References

1 Manuel Bodirsky, Jan Kára, and Barnaby Martin. The complexity of surjective homomorphism problems - a survey. *Discrete Applied Mathematics*, 160(12):1680–1690, 2012.

2 Flavia Bonomo, Gianpaolo Oriolo, and Claudia Snels. Minimum weighted clique cover on strip-composed perfect graphs. In *Proc. WG 2012*, volume 7551 of *Lecture Notes in Computer Science*, pages 22–33, 2012.

3 Andries E. Brouwer and Henk Jan Veldman. Contractibility and NP-completeness. *Journal of Graph Theory*, 11(1):71–79, 1987.

4 Danny Z. Chen, D. T. Lee, R. Sridhar, and Chandra N. Sekharan. Solving the all-pair shortest path query problem on interval and circular-arc graphs. *Networks*, 31(4):249–258, 1998.

5 Maria Chudnovsky and Paul D. Seymour. *The structure of claw-free graphs*, volume 327 of *London Mathematical Society Lecture Note Series*, pages 153–171. Cambridge University Press, 2005.

6 Kathryn Cook, Simone Dantas, Elaine M. Eschen, Luérbio Faria, Celina M. H. de Figueiredo, and Sulamita Klein. $2K_2$ vertex-set partition into nonempty parts. *Discrete Mathematics*, 310(6-7):1259–1264, 2010.

7 Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. A polynomial algorithm for 3-compatible coloring and the stubborn list partition problem (the stubborn problem is stubborn no more). *SIAM Journal on Computing*, 41(4):815–828, 2012.

8 Simone Dantas, Celina M. H. de Figueiredo, Sylvain Gravier, and Sulamita Klein. Finding H-partitions efficiently. *RAIRO - Theoretical Informatics and Applications*, 39(1):133–144, 2005.

9 Simone Dantas, Frédéric Maffray, and Ana Silva. $2K_2$-partition of some classes of graphs. *Discrete Applied Mathematics*, 160(18):2662–2668, 2012.

10 Celina M. H. de Figueiredo. The P versus NP-complete dichotomy of some challenging problems in graph theory. *Discrete Applied Mathematics*, 160(18):2681–2693, 2012.

11 Yuri Faenza, Gianpaolo Oriolo, and Gautier Stauffer. Solving the weighted stable set problem in claw-free graphs via decomposition. *Journal of the ACM*, 61(4):20:1–20:41, 2014.

12 Tomás Feder and Pavol Hell. List homomorphisms to reflexive graphs. *Journal of Combinatorial Theory, Series B*, 72(2):236–250, 1998.

13 Tomás Feder, Pavol Hell, Sulamita Klein, and Rajeev Motwani. List partitions. *SIAM Journal on Discrete Mathematics*, 16(3):449–478, 2003.

14 Jirí Fiala, Marcin Kaminski, and Daniël Paulusma. A note on contracting claw-free graphs. *Discrete Mathematics & Theoretical Computer Science*, 15(2):223–232, 2013.

15 Herbert Fleischner, Egbert Mujuni, Daniël Paulusma, and Stefan Szeider. Covering graphs with few complete bipartite subgraphs. *Theoretical Computer Science*, 410(21-23):2045–2053, 2009.

16 Petr A. Golovach, Matthew Johnson, Barnaby Martin, Daniël Paulusma, and Anthony Stewart. Surjective H-colouring: New hardness results. *Computability*, to appear.

17 Frank Harary. *Graph Theory*. Reading MA, 1969.

18 Pavol Hell. Graph partitions with prescribed patterns. *European Journal of Combinatorics*, 35:335–353, 2014.

19 Danny Hermelin, Matthias Mnich, Erik Jan van Leeuwen, and Gerhard J. Woeginger. Domination when the stars are out. *CoRR*, abs/1012.0012, 2010.

20 Danny Hermelin, Matthias Mnich, Erik Jan van Leeuwen, and Gerhard J. Woeginger. Domination when the stars are out. In *Proc. ICALP 2011*, volume 6755 of *Lecture Notes in Computer Science*, pages 462–473, 2011.

21 Takehiro Ito, Marcin Kaminski, Daniël Paulusma, and Dimitrios M. Thilikos. On disconnected cuts and separators. *Discrete Applied Mathematics*, 159(13):1345–1351, 2011.

22 Takehiro Ito, Marcin Kaminski, Daniël Paulusma, and Dimitrios M. Thilikos. Parameterizing cut sets in a graph by the number of their components. *Theoretical Computer Science*, 412(45):6340–6350, 2011.

23 Marcin Kaminski, Daniël Paulusma, Anthony Stewart, and Dimitrios M. Thilikos. Minimal disconnected cuts in planar graphs. *Networks*, 68(4):250–259, 2016.

24 Andrew D. King. *Claw-free graphs and two conjectures on ω, Δ, and χ.* PhD thesis, University, Montreal, Canada, 2009.

25 Andrew D. King and Bruce A. Reed. Bounding χ in terms of ω and δ for quasi-line graphs. *Journal of Graph Theory*, 59:215–228, 2008.

26 Barnaby Martin and Daniël Paulusma. The computational complexity of disconnected cut and $2K_2$-partition. *Journal of Combinatorial Theory, Series B*, 111:17–37, 2015 (conference version Proc. CP 2011, LNCS 6876, 561–575).

27 Ross M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2):93–147, 2003.

28 Rafael B. Teixeira, Simone Dantas, and Celina M. H. de Figueiredo. The external constraint 4 nonempty part sandwich problem. *Discrete Applied Mathematics*, 159(7):661–673, 2011.

29 Narayan Vikas. Computational complexity of compaction to reflexive cycles. *SIAM Journal on Computing*, 32(1):253–280, 2002.

30 Narayan Vikas. Algorithms for partition of some class of graphs under compaction and vertex-compaction. *Algorithmica*, 67(2):180–206, 2013 (conference version Proc. COCOON 2011, LNCS 6842, 319–330).

Practical Low-Dimensional Halfspace Range Space Sampling

Michael Matheny
University of Utah, USA

Jeff M. Phillips[1]
University of Utah, USA

Abstract

We develop, analyze, implement, and compare new algorithms for creating ε-samples of range spaces defined by halfspaces which have size sub-quadratic in $1/\varepsilon$, and have runtime linear in the input size and near-quadratic in $1/\varepsilon$. The key to our solution is an efficient construction of partition trees. Despite not requiring any techniques developed after the early 1990s, apparently such a result was never explicitly described. We demonstrate that our implementations, including new implementations of several variants of partition trees, do indeed run in time linear in the input, appear to run linear in output size, and observe smaller error for the same size sample compared to the ubiquitous random sample (which requires size quadratic in $1/\varepsilon$). This result has direct applications in speeding up discrepancy evaluation, approximate range counting, and spatial anomaly detection.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Partitions, Range Spaces, Sampling, Halfspaces

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.62

1 Introduction

Taming the relationship between a point set $X \subset \mathbb{R}^d$ and its interaction with halfspaces $\mathcal{H}_d$, has long been a focus of computational geometry. Understanding and controlling this interaction is at the heart of problems in range searching, linear classification, coresets, and spatial anomaly detection. This pair $(X, \mathcal{H}_d)$ describes a range space, the combinatorial set of all subsets of X defined by $h \cap X$ for any halfspace $h \in \mathcal{H}_d$. In this paper we focus on two specific and closely-interrelated (as it turns out) constructions for $(X, \mathcal{H}_d)$: ε-samples and partitions, defined next.

An *ε-sample* $Y \subset X$ of $(X, \mathcal{H}_d)$ is a small point set that approximately preserves density with respect to halfspaces: for all $h \in \mathcal{H}_d$, and error parameter $\varepsilon \in (0,1)$ it bounds

$$\mathsf{Error}(X,Y) = \max_{h \in \mathcal{H}_d} \left| \frac{|Y \cap h|}{|Y|} - \frac{|X \cap h|}{|X|} \right| \leq \varepsilon.$$

It is known that ε-samples of size $\Theta(1/\varepsilon^{2d/(d+1)})$ exist for halfspaces [3], and in general this size may be required [21]. For many years (c.f., [22, 8]) such proofs were not constructive, as they relied on the "partial coloring lemma"; until in 2010 when Bansal [5] introduced a polynomial time construction. The runtime of the low-discrepancy coloring on m points was later reduced [16] to $O(m^{3(d+1)}\text{polylog}(m))$, this within the standard merge-reduce framework [10] results in a $O(n(1/\varepsilon)^{2d(3d+2)/(d+1)}\text{polylog}(1/\varepsilon))$ runtime for sample construction– which is

[1] Thanks to supported by NSF CCF-1350888, IIS-1251019, ACI-1443046, CNS-1514520, and CNS-1564287.

© Michael Matheny and Jeff M. Phillips;
licensed under Creative Commons License CC-BY

still not very efficient. For instance for $d = 2$, this requires $O(n(1/\varepsilon)^{10+2/3}\text{polylog}(1/\varepsilon))$ time. A random sample, which can be generated in $O(n + 1/\varepsilon^2)$ time, is an ε-sample of size $O(\frac{1}{\varepsilon^2}(d + \log \frac{1}{\delta}))$ with probability at least $1 - \delta$ [27, 15]. The above discrepancy-based algorithm can be run on the output of this sample to get optimal size, but it only reduces the overall runtime of the ε-sample construction to $O(n + (1/\varepsilon)^{2d(3d+2)/(d+1)+2}\text{polylog}(1/\varepsilon))$.

There are other constructions for ε-samples, which either focus on small space (to work in a stream) [26, 4] or have better performance in practice without size guarantees below that of random sampling [2]. As with the optimal algorithms, these require the enumeration of all combinatorial halfspaces associated with a set of size roughly the size of the final ε-sample, requiring at least $\Omega((1/\varepsilon^{2d/(d+1)})^d)$ time. Indeed Suri *et al.* [26] concludes with: "*The high computational complexity of the currently known algorithms for these subroutines may be prohibitive for data stream applications. It is a long standing open problem to find efficient exact or approximation algorithms for either of them.*"

A *partition* of $(X, \mathcal{H}_d)$ is a set of pairs $\{(\Delta_1, X_1), (\Delta_2, X_2), \ldots\}$ where each Δ_i is a small complexity region and contains $X_i \subset X$, and X is the disjoint union of the X_is. It is a (t, z)-*partition* when there are $O(t)$ pairs, $|X_i| \leq 2n/t$; and each $h \in \mathcal{H}_d$ crosses $O(t^z)$ cells. The smallest possible guarantee for z is $z = (1 - 1/d)$, and an algorithm for such a construction was provided by Matoušek [20], that takes $O(n \log t)$ time after $O(n^{1+\eta})$ preprocessing time for any $\eta > 0$. Chan provided a refined algorithm which takes $O(n \log t)$ time, and has a few nicer structural properties. There are other algorithms which generate (t, z)-partitions for large values of z. For instance in $\mathbb{R}^2$ Edelsbrunner and Welzl [11] describe an algorithm with $z = 0.695$ and a structure similar to a kd-tree leads to a size of $z = \log_4(3) \leq 0.7925$ [29].

Our results. In this paper, we use partition construction algorithms to efficiently create ε-samples for $(X, \mathcal{H}_d)$. Our algorithm takes $O(n + \frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon})$ time and produces an ε-sample of size $O((1/\varepsilon)^{2d/(d+1)} \log^{d/(d+1)}(1/\varepsilon))$, nearly matching the $\Omega(1/\varepsilon^{2d/(d+1)})$ lower bound.

We also implement several variants of these algorithms in $\mathbb{R}^2$. We know of no other implementation of ε-sample construction for $(X, \mathcal{H}_2)$ which is guaranteed to get subquadratic size in $1/\varepsilon$. We know of no implementations of optimal partitions, although Har-Peled [13] has implemented a related concept called a cutting, which (as we will explain) is a key ingredient for creating partitions. We choose to build our own implementation of cuttings, and explain why we did not use Har-Peled's in Section 4.

We are able to demonstrate that our algorithm indeed scales linearly in n, scales linearly in the output size, and produces ε-samples with less measured error than random samples.

Our initial goal in fast ε-sample construction comes from finding approximately maximal ranges in range spaces, as part of a large-scale spatial anomaly detection framework [18, 17]. At a high level, these algorithms follow two phases: (1) create an ε-sample S, (2) use S to find an approximately maximal range. The second step takes $O(|S|/\varepsilon)$ or $O(|S|/\varepsilon^2)$, so it is only worth using a smaller ε-sample of size roughly $1/\varepsilon^{4/3}$ if it takes less than $1/\varepsilon^{2+1/3}$ or $1/\varepsilon^{3+1/3}$ time to create. We show this is the case in theory, and in practice. Similar overall runtime gains exist when using S for classification, or approximate range counting, or other tasks where the use of S is more expensive than the new construction time.

2 Overview and Proof for Fast ε-Samples

The key to our construction of an ε-sample S for a range space $(X, \mathcal{H}_d)$ is to first create a partition over $(X, \mathcal{H}_d)$. Given such a partition algorithm, our algorithm constructs an ε-sample as follows. Randomly sample $Y \subset X$, construct the partition $\Delta = \{(\Delta_1, Y_1), \ldots, \}$ on Y, and return a single point at random from each Y_i weighted by $|Y_i|$.

▶ **Theorem 1.** *For range space $(X, \mathcal{H}_d)$ with $|X| = n$ and constant d, with constant probability an ε-sample S of size $O(\frac{1}{\varepsilon^{2d/(d+1)}} \log^{d/(d+1)} \frac{1}{\varepsilon})$ can be constructed in $O(n + \frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon})$ time.*

Proof. Take a random uniform sample $Y \subset X$ of size $s = O(\frac{1}{\varepsilon_1^2})$ then Y is an ε_1-sample of $(X, \mathcal{H}_d)$ with constant probability. Next we build a $(t, 1 - 1/d)$-partition on Y in $O(s \log t)$ time [7]; this results in a set of $O(t)$ partitions of Y each containing at most $2s/t$ points such that any halfspace in $\mathcal{H}_d$ will only cross $O(t^{1-1/d})$ of them. From each partition (Δ, Y_i) we will choose a single point y_i at random to put in our result S, and weight it proportional to the number of points in the partition.

In our construction any partition contained completely inside a halfspace or outside does not contribute to the error of the sample. Only regions crossing the boundary of the halfspace h contribute to the error. The error in each boundary region is an independent bounded random variable V_i with value in the range $[0, 2\frac{s}{t}]$. There are at most $k = c \cdot t^{1-1/d}$ boundary regions for some constant c, so we can apply Hoeffding's inequality, with failure probability δ

$$\Pr[|V - \mathsf{E}[V]| \geq s\varepsilon_2] \leq 2 \exp\left(-\frac{2s^2\varepsilon_2^2}{ct^{1-1/d} \cdot 4\frac{s^2}{t^2}}\right) = 2 \exp\left(-\frac{\varepsilon_2^2 t^{1+1/d}}{2c}\right) \leq \delta.$$

Rearranging the last inequality, gives that with $t \geq (\frac{2c}{\varepsilon_2^2} \ln \frac{2}{\delta})^{d/(d+1)}$, for any one halfspace h, $|V - \mathsf{E}[V]|$ is more than $s\varepsilon_2$ with probability at most δ.

There are $O(s^d) = O(1/\varepsilon_1^{2d})$ halfspaces in $(Y, \mathcal{H}_2)$, so setting $\delta = c_2\varepsilon_1^{2d}$ for some constant c_2, and the additivity property of ε-approximations [8], gives an $(\varepsilon_1 + \varepsilon_2)$-approximation of size $t \geq \left(\frac{4dc}{\varepsilon_2^2} \ln \frac{2}{c_2\varepsilon_1}\right)^{d/(d+1)}$ with constant probability. By setting $\varepsilon_1 = \varepsilon_2 = \frac{\varepsilon}{2}$ the total error is $\varepsilon_1 + \varepsilon_2 = \varepsilon$ and the size of the ε-sample is $O\left(\frac{1}{\varepsilon^{2d/(d+1)}} \log^{d/(d+1)}\left(\frac{1}{\varepsilon}\right)\right)$ for constant d. Creating Y takes $O(n + \frac{1}{\varepsilon^2})$ time, the partition tree construction takes $O(\frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon})$ time since $t = O(\mathsf{poly}(\frac{1}{\varepsilon}))$, and the re-weighting and sampling step takes $O(\frac{1}{\varepsilon^2})$ time. In total therefore the entire algorithm takes $O(n + \frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon})$ time. ◀

The same proof technique will work with other (t, z)-partitions in place of Chan's [7]. In general, for $z < 1$, a scheme that generates a (t, z)-partition of t cells where any halfspace crosses at most $O(t^z)$ of the cells results in an ε-sample of size $O(\frac{1}{\varepsilon^{2/(2-z)}} \log^{1/(2-z)} \frac{1}{\varepsilon})$. For instance in $\mathbb{R}^2$, Edelsbrunner and Welzl's $z = 0.695$ result [11] in an ε-sample of size $O(\frac{1}{\varepsilon^{1.532}} \log^{0.766}(\frac{1}{\varepsilon}))$. Alternatively, Willards $z = 0.7925$ result in $\mathbb{R}^2$ [29] results in an ε-sample size of $O(\frac{1}{\varepsilon^{1.657}} \log^{0.829}(\frac{1}{\varepsilon}))$.

3 Overview of Algorithms for Constructing the Partition

Random sampling, and sampling a point from each cell of a partition is straight-forward; the challenge in our implementation of Theorem 1 is the creation of a partition. In this section we describe the key components of the two prominent optimal size $(z = 1 - 1/d)$ algorithms: Matoušek's efficient partitioning [20] (ComputePartition-Mat) and Chan's Optimal partitioning [7] (ComputePartition-Chan).

These algorithms rely on a related object called a cutting, defined over $\mathbb{R}^d$ and a set of m hyperplanes H. For a parameter $r < m$, a $(1/r)$-*cutting* is a decomposition of $\mathbb{R}^d$ into $O(r^d)$ cells $\Lambda = \{\Lambda_1, \Lambda_2, \ldots\}$, so no cell is crossed by more than $O(m/r)$ hyperplanes in H. Such cuttings exist and can be computed in $O(mr^{d-1})$ time [9, 19].

Cuttings are almost enough to compute partitions. A set of n points in $\mathbb{R}^d$ induces $m = O(n^d)$ combinatorially distinct halfspaces H. Letting $r = t^{1/d}$, the total number of crossings will be $O(r^d \cdot m/r) = O(mr^{d-1})$, so the *average* per region will be $O(r^{d-1}) = O(t^{1-1/d})$. Also, ignoring dependences, the average cell contains $O(n/r^d) = O(n/t)$ points, as desired. The main challenge is ensuring that these average properties of the cutting map to the specific properties required for the partition. In short, we can create an appropriate cutting, detect where it does not satisfy the partition properties, and then amend it so it does.

We specifically focus our implementations in the $d = 2$ setting, which for instance is enough for our original application of spatial anomaly detection we mentioned previously [17], even in higher dimensions. Our implementations are similar to the existing implementation of cuttings by Har-Peled [13], but adds several features which will aid in computing the partition. Our cutting implementation builds a cutting by iteratively adding lines in a random order while keeping track of the number of lines crossing each cell in an arrangement. From a practical point of view, it is important to force the cells of the partition to be constant size. We have focused on two methods for this, a vertical trapezoidal decomposition (Trapezoid), or a hierarchy of constant size polygons (PolyTree).

Constructing a $(1/r)$-cutting over the entire set of $O(n^d)$ halfspaces would lead to a runtime of $O(n^d r^{d-1})$ which would be prohibitively slow. Instead of using the full set of halfspaces a smaller set (a test set) can be constructed, such that the number of partitions crossed by any halfspace in this test set will not be too different from the full set $\mathcal{H}_d$.

In particular, an $(1/r)$-*test set* is a set of halfspaces H which applies to any partition $\Delta = \{(\Delta_1, X_1), (\Delta_2, X_2), \ldots\}$ and point set X of size n so $|X_i| \geq n/r$ for all $(\Delta_i, X_i) \in \Delta$. It ensures that if $\kappa = \max_{h \in H} |h \cap \Delta|$, then $\max_{h \in \mathcal{H}_d} |h \cap \Delta| \leq O(\kappa + r^{1-1/d})$. Here $h \cap \Delta$ is the set of $(\Delta_i, X_i) \in \Delta$ for which Δ_i intersects h, but do not completely contain h. Test sets can be built a number of ways, including randomly sampling lines, randomly sampling points and using the lines they induce, and using the dual arrangement.

4 Implementation Particulars of Partitions

Our implementation of Partition trees is in python. It relies on an efficient way to construct and maintain an arrangement of lines and associated points. At each step of the construction we will maintain a tree with leaves that correspond to cells $\Delta_1, \Delta_2, \ldots$ of an arrangement. Each cell will maintain a list of contained points $X_i \in \Delta_i$ and crossing lines.

As part of the construction so the result is a $(t, 1 - 1/d)$-partition Δ, with desired t parameter, cells can be refined by applying various operations to them. For instance a cutting can be constructed locally inside of a cell Δ_i, or a cell can be partitioned into a set of sub-cells.

Geometric Primitives. All of our algorithms rely on operations over line segments. The most important operation is being able to test, within a region Δ, if a line lies completely above a line segment or if it crosses a line segment. This fairly simple operation is slightly complicated by numerical issues that can occur. For instance when constructing a test set using the BuildTestSet-Points or BuildTestSet-Dual method (see below) many lines will potentially meet at the same point. Line segments that meet in this point could be mistaken as crossing. To handle numerical issues we use python's implementation of `math.isclose` to handle point comparisons. This method allows us to assign two floating point numbers as equal if their relative values are sufficiently close [6]. Moreover, all methods that compare line segments have closed and open versions where closed versions allow end-point overlap and open versions do not. The method `segment.above_closed(line)` returns true if the `line` intersects with the segment at one the segment's end points, but is otherwise above the segment, while `segment.above_open(line)` returns false in this case. This allows us in our experiments to effectively handle degeneracies while avoiding slower exact precision libraries.

Internally our segment objects are represented by the slope, a, the y-intercept, b, and the $[x_l, x_r]$ interval on the x-axis the segment is defined over. This representation makes many operations easy, but also results in several challenges, most notably: vertical lines are undefined, unbounded segments (e.g., $(-\infty, x_r]$) require extra logic to handle crossing queries, lines which are nearly vertical can become numerically unstable, and the dual of unbounded polygons require significant extra logic to handle correctly. However, we have implemented stable functions for intersect and above relations for pairs of segments in a cell.

Using line segments and points as the primitives we also define more complicated structures notably: polygons, dual wedges, vertical line segments, and trapezoids.

PolyTrees. There are a number of ways to maintain the structure of an arrangement. A common method is to store each cell with a corresponding list of pointers to adjacent cells. Inserting a line involves finding the leftmost crossing cell, identifying the next adjacent cell the line crosses, splitting the crossed cell into an upper and lower cell, and then repeating this operation for each crossed cell.

This has a number of downsides: there are special cases if a line crosses a vertex of a cell, inserting points into the arrangement requires the maintenance of a secondary structure, and cells require a significant amount of adjacency information that must be maintained. Instead of maintaining this structure we use the idea of forcing each cell to be simple, and follow certain restrictions, as introduced by Seidel [25] and refined by Har-Peled [13].

In particular, we either maintain a decomposition into constant complexity polygons (polygons with a constant number of boundary segments) or a trapezoidal decomposition. In both cases we maintain a tree where each node in the tree consists of a line segment that separates a cell into two cells. With trapezoids an inserted line could in some cases divide a trapezoid vertically into 2 separate trapezoids and then horizontally into 4 separate trapezoids. In the case of polygons the inserted line would split the polygon into two separate polygons which could possibly be further split if the number of sides in either of the resulting polygons is greater than a chosen constant. We also enforce that no vertical segments are used to avoid limitations of our line segment representation.

Given a line h and a decomposition $\Lambda = \{\Lambda_1, \Lambda_2, \ldots\}$, the *zone* of h is the set of regions Λ_i that intersect h; we represent this as $\mathsf{Zone}_h = \Lambda \cap h$. To find the zone of a line in this structure at each node we treat the line as an infinite length segment and then traverse the line down the tree. At each node we will have three cases where the portion of the line contained in the node lies completely either above or below, or crosses the current node's line segment. In the completely above or below case we merely traverse to the above or below child of the node. In the crossing case we split the portion of the line contained in the node into two segments, above and below, and recursively query the above and below nodes. Point information is easy to maintain with this method since a point always lies on one side of the line segment, so the tree structure can be used to insert or remove points in logarithmic time to the number of cells.

More complicated structures can also be queried on these trees, most notably wedges and polygons. Wedge queries are particularly useful in ComputePartition-Chan since a wedge is the dual of a line segment, so the number of points contained in a wedge corresponds in the dual to the number of lines crossing a line segment.

Cuttings. Our cutting algorithm CreateCutting(H, r) (Algorithm 1) follows closely Algorithm 1, from Har-Peled [13]. We implement the cutting with respect to weighted lines as this speeds up and somewhat simplifies the later partitioning algorithms. We require a

Algorithm 1 CreateCutting(H, r).

```
1: $\Lambda = \mathbb{R}^2$
2: for $h \in H$ (ordered by a random weighted permutation) do
3:     Find $\mathsf{Viol}(h, \Lambda) = \{\Lambda_i \in \mathsf{Zone}_h(\Lambda) \mid |H \cap \Lambda_i| > |H|/r\}$.
4:     For all $\Lambda_i \in \mathsf{Viol}(h, \Lambda)$, replace $\Lambda_i$ in $\Lambda$ by $\mathsf{split}(\Lambda_i, h)$
5: return $\Lambda$
```

Algorithm 2 BuildTestSet-Dual (X, r).

```
1: $S = \text{sample}(X, O(\sqrt{r}\log r))$; $S^*$ is dual of $S$.
2: $\Lambda \leftarrow \mathsf{CreateCutting}(S^*, O(r^{1/2}))$
3: return the dual of $V^*$, where $V$ is the set of vertices of the cells of $\Lambda$.
```

weighted permutation of lines using [12]; this ensures that the probability we see a line after some point in the permutation is equivalent to the probability we would have seen at least one instance after seeing that many distinct lines in a variant where weights are multiplicities (as advocated by Chan [7]), and each copy is treated independently in a uniform random permutation. For notational convenience, for a subset $H' \subseteq H$, let $|H'| = \sum_{h \in H'} w(h)$, where $w(h)$ is the weight implicitly stored with each halfspace $h \in H$.

In practice, we implement CreateCutting(H, r) slightly differently then described in Algorithm 1. Instead of choosing the $h \in H$ to process in the random weighted permutation, our approach is centered around the violated cells. We choose a cell $\Lambda_i \in \Lambda$ which is too heavy (i.e., $|H \cap \Lambda_i| > |H|/r$), and then choose some halfspace $h \in \Lambda_i \cap H$, and only replace Λ_i (not the entire Zone$_h$) with the result of the split(Λ_i, h), which divides Λ_i into two parts separated by h. This change has two advantages. First we do not need to find the Zone$_h(\Lambda)$ which involves traversing the PolyTree, so our approach is slightly faster. Second, we can choose the $h \in \Lambda_i \cap H$ to use in the split wisely; e.g., as the one that maximizes the smaller of the two resulting cells. We find the second heuristic produces slightly smaller cuttings in practice, but is significantly slower, and is not used in our experiments.

How split(Λ_i, h) is implemented is the difference between the Trapezoid-based cutting and the Polygon-based cutting we refer to as PolyTree.

For the Trapezoid-based method each cell Δ is a trapezoid to begin with. The split operation first inserts up to two new vertical cuts for each intersection of the line with the top or bottom of the cell and then horizontally cuts the resulting cells using the inserted line. For PolyTree, the first important detail is that we store $H \cap \Lambda_i$ as the line segments restricted to where they intersect Λ_i.

On a split, we need to maintain which halfspaces $h \in \Lambda_i \cap H$ are in each child; if $h' \in \Lambda_i \cap H$ intersects both children, then we split h' into two line segments at the point where it intersects h, and store the corresponding segment in each child. If $h' \in \Lambda_i \cap H$ is in only one child, because we store them as segments it is easy to check which child it goes into.

Both of these algorithms are then fairly straightforward to implement once given structures for efficiently maintaining arrangements of line segments.

We find that PolyTree is faster and produces a smaller cutting than Trapezoid-based ones; see Figure 1 and Figure 2 which show the runtime and cutting size as a function of input size and choice of r. For this reason we will primarily focus on PolyTree hereafter.

Test Set Generation. There are a number of ways to generate test sets (BuildTestSet-Dual, BuildTestSet-Points, BuildTestSet-Lines), but these do not appear to have a significant effect on the runtime of the final algorithms; again see Figure 1 and Figure 2 for comparisons

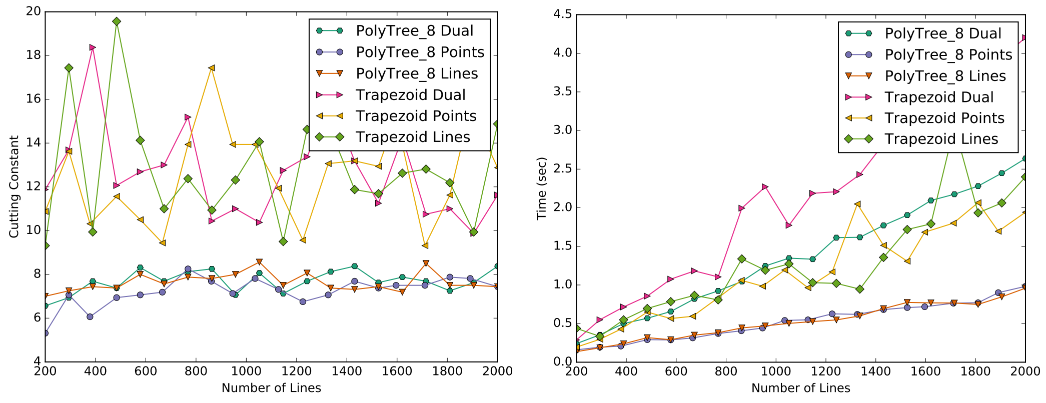


Figure 1 Size of cutting (divided by r^2) and time (in seconds) vs. the number of input lines.

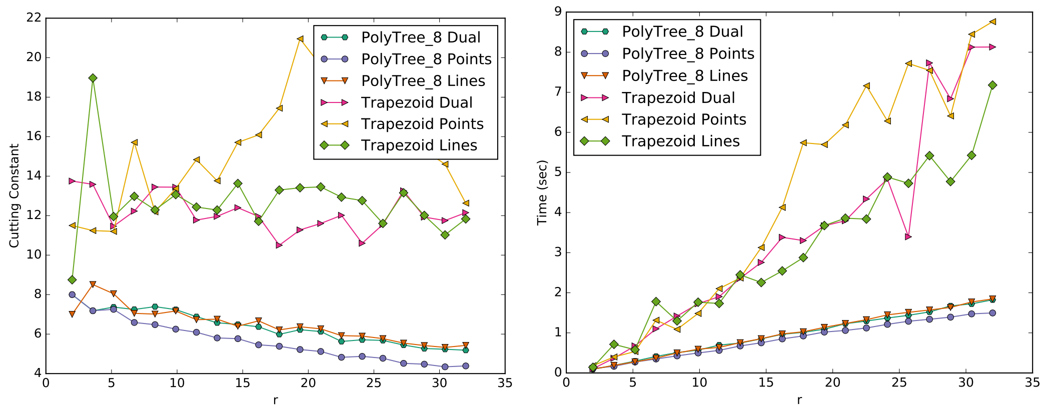


Figure 2 Size of cutting (divided by r^2) and time (in seconds) as r increases for 1000 input lines.

of the PolyTree and Trapezoid methods. The simplest method, BuildTestSet-Lines, simply samples $O(r \log^d n)$ halfplanes, from those defined by passing through d points in X. The next simplest, BuildTestSet-Points, samples $O(r^{1/d} \log n)$ points S, and then the test set is all halfplanes passing through d-tuples chosen from S; it again defines $O(r \log^d n)$ halfplanes. Finally the most complicated approach is BuildTestSet-Dual (see Algorithm 2); it produces the smallest size test set, size $O(r)$ [20], and thus is the one we advocate. It samples $O(r^{1/d} \log r)$ points $S \subset X$; it considers the dual set of halfplanes S^* of primal points S; it creates a $(1/r^{1/d})$-cutting of S^* (in the dual); and then it returns the primal halfspaces defined by the vertices of the cutting in the dual. Each halfspace h in the test set H is implicitly endowed with a weight $w(h)$, which by default is $w(h) = 1$ for all $h \in H$.

Matoušek Partitioning. We have implemented Matoušek's efficient partition trees [20]. At a high level this algorithm computes the cutting of a test set and then finds a single good cell that contain at least n/b points (for a constant b, we use $b = 16$ as default). It adds this cell to the partition, doubles the weight of all halfspaces in the test set crossing that cell, computes a new cutting and good cell. It repeats until the number of points remaining has been cut by half, and then it recurses on the remained of the points at half the precision (e.g., set $b := 1/2b$). This is too expensive to do with $b = r$, so after this we then recursively partition each cell (Δ_i, X_i) until the result is an $(1/r, 1 - 1/d)$-partition

Algorithm 3 ComputePartition-Mat(X, r, n, j).

```
1: if $(|X| < n/r)$ then return $\{(X, \Delta_0)\}$ where $\Delta_0$ contains $X$.
2: $H \leftarrow$ BuildTestSet-x$(X, b/2^j)$
3: $\Delta = \emptyset$
4: while $(|X| \geq n/2^j)$ do
5:     $\Lambda \leftarrow$ CreateCutting$(H, \sqrt{b/2^j})$
6:     Find $\Lambda_i \in \Lambda$ so $|X \cap \Lambda_i| > n/b$; shrink $\Lambda_i$ so $|X \cap \Lambda_i| = \lfloor n/b \rfloor$ exactly.
7:     Add $(\Lambda_i, \Lambda_i \cap X)$ to $\Delta$; remove $\Lambda_i \cap X$ from $X$
8:     Double the weight $h \in H$ which cross $\Lambda_i$
9: $\Delta' = \bigcup_{(\Delta_j, X_j) \in \Delta}$ ComputePartition-Mat$(X_j, r, n, j)$
10: return $\Delta' \cup$ ComputePartition-Mat$(X, r, n, j+1)$
```

Algorithm 4 ComputePartition-Chan(Δ, r, n).

```
1: Trim to $\Delta' = \{(\Delta_i, X_i) \in \Delta \mid |X_i| > n/r\}$; if $\Delta' = \emptyset$ return $\Delta$
2: $H =$ BuildTestSet-x$(X, |\Delta|)$
3: for $(\Delta_i, X_i) \in \Delta'$ do
4:     Sample $L \subset H$, proportional to their weight $w(h)$, at rate $q$
5:     $\Lambda =$ CreateCutting$(L, r_i)$; with $r_i$ chosen so $|\Lambda| \leq b/4$
6:     For all $\Lambda_j \in \Lambda$, further split $\Lambda_j$ (with split) until $|X_i \cap \Lambda_j| \leq |X_i|/b$
7:     Replace $(\Delta_i, X_i)$ in $\Delta$ with $\{(\Lambda_1, \Lambda_1 \cap X_i), (\Lambda_2, \Lambda_2 \cap X_i), \ldots\}$
8:     Update all weights $w(h) = w(h)(1 + 1/b)^{|h \cap \bar{\Lambda}_i|/p}$.
9: return ComputePartition-Chan$(\Delta, r, n)$
```

as desired. The branching factor of the partition tree is not fixed on each level, but will be roughly b on average. Algorithm 3 presents this approach, and is initially called as ComputePartition-Mat$(X, r, |X|, 0)$.

Note that Line 9 is the refinement step where each cell is further partitioned. Since the first level is the most important for good ε-samples, faster algorithms could be used at later recurve calls at this step. In contrast, the recursive call at Line 10 is handing objects not handled in the first pass, where each pass handles roughly half of the data.

Chan Partitioning. Chan's optimal partition trees [7] are faster in theory than Matoušek's algorithm, but are more complicated to implement. The algorithm works by processing each node at a certain level in the tree in a random order. For each node it creates a cutting of approximately $b/4$ size for an appropriately large branching parameter b (our implementation uses $b = 22$ as a default). It then further splits the cells of the cutting to contain $1/b$ fraction of points at that node. It multiplicatively updates weights for halfplanes that cross each cell. This multiplicative update influences subsequent cuttings by biasing away from creating cells that are crossed by already heavily weighted lines (lines that cross many cells). After splitting all of the cells in this level of the tree the algorithm recurses on the newly created level. The ultimate partition $\Delta = \{(\Delta_1, X_1), (\Delta_2, X_2), \ldots\}$ are the leaf nodes of the tree. Algorithm 4 presents this approach, calling ComputePartition-Chan$(\{(\mathbb{R}^2, X)\}, r, |X|)$ initially.

Implementing the algorithm as described is too slow asymptotically, so Chan presents a faster variant, which requires two additional parameters p and q. Roughly $q = \sqrt{b|\Delta|}/|X|$ (see [7] for details) determines the probability that a line ends up in the reduced test set L. The parameter p, about $\sqrt{b/|\Delta|}\log n$ (again, see [7] for details), effects the number of cells Λ_i that are used to update the weight in each h (we sample each cells with probability p as opposed to dividing by this number, as written on Line 8). Also, Line 4, where L is sampled from H, can be made more efficient by only minimally updating L each pass through the loop, since it generally has large weight lines and that set is fairly stable.

Near the bottom of the tree, Line 8 can be expensive. We make this efficient with a crucial observation that the test set H was generated by computing a cutting over the dual space. Thus these halfspaces are duals to the vertices of the PolyTree structure. Thus we can search over the PolyTree to determine the number of crossing lines. A cell of the partitioning is a polygon consisting of a constant number of line segments. A line crossing the polygon will cross at least one of the line segments and in the dual this will correspond to a point contained inside of a double wedge. For each line segment in the polygon we take its dual (a double wedge) and query the PolyTree that was used to construct the test set to determine the number of vertices contained inside of it. Since we only return the overlapped polygons and each polygon consists of at most a constant number of edges, the number of queried cells can only be a constant factor larger than the number of lines crossed by the line segment.

However, code profiling shows that the two steps involving sampling with p and q, and updating L are the most expensive parts of the algorithm. As a result we also consider a variant ComputePartition-Chan-Simple which avoids these sampling steps that were supposed to speed things up. In the context of Algorithm 4 this basically sets $p = q = 1$, so $L = H$, and Line 4 is not required.

The given algorithm is only guaranteed to compute a set of partitions in $O(n \log^{O(1)} n)$ time; incurring extra log factors due to the height of the partition tree. Chan removes log factors with a method he calls bootstrapping. We do not do this since the branching factor is high (around 22) so the depth of the tree is low, and this method is not worth the overhead.

In our implementation, we only compute the test set H once at the beginning. On each recursive call (Line 9) we can reuse it, but simply reset all of the weights to be uniform.

Ham-Sandwich Tree. We also implement an alternative using Willard's [29] Ham-Sandwich Tree. It provides a partitioning with $z = \log_3 4$, which gives a $O(\frac{1}{\varepsilon^{1.657}} \log^{0.829} \frac{1}{\varepsilon})$ sized sample, and constructs a tree with a branching factor of 4. At each level we split the point set in half with a single vertical line, and on these two resulting sets we find a single (roughly horizontal) line that divides both the left and right point set in half. Such a separator is guaranteed by the ham-sandwich theorem, and can be computed in linear time [24], but is complicated to implement. We instead approximate the ham-sandwich cut by computing a number of test lines and choosing the best separator from these. This is simple to implement, gives good cuts in practice, and can guarantee to be at most ε-imbalanced [23].

Why not use Har-Peled's implementation and CGAL? It may seem at first that we could simply use Har-Peled's implementation for ε-cuttings [13]. However, our initial goal was to use this as part of a code for spatial anomaly detection [18, 17], and there were several issues that made this less feasible. (1) We wanted to use non infinite precision floating point arithmetic. Har-Peled reports that switching to exact precision representations results in a 30-factor slow down, but was necessary for degeneracy issues. We managed these precision issues while using floating point arithmetic with careful use of open and closed operators for line above/below and intersection. (2) We can measure wedges, and line segments on the PolyTree structure which is very useful in ComputePartition-Chan. (3) Har-Peled's code created a cutting inside of a 1×1 box. This makes computing dual cuttings difficult as we first have to normalize the lines to lie in such a region, but computing the correct normalization quickly would require us to re-implement much of the PolyTree algorithm. Ultimately we opted to build our ε-cutting code from scratch rather than modify the previous code.

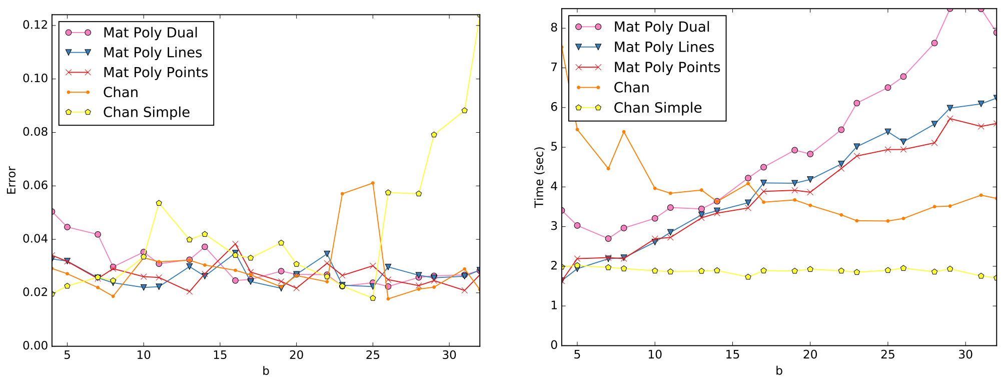


Figure 3 The Branching Factor b vs. Time and Error using the default parameters.

Har-Peled [13] also reported the cutting constant (number of cells divided by r^2) for various of his algorithms, about 7.3 (polygons) and 12.8 (trapezoids). This roughly matches the numbers we observe in Figure 1 and Figure 2.

Another option for computing cuttings and managing the partition trees is using the current 2d-arrangement implementation in CGAL [28]. This would have most likely made portions of this project much easier to implement and removed various hurdles. However, the possibilities of several factor slow-downs using exact precision would have potentially resulted in no ultimate gains in the spatial anomaly application demonstrated below.

5 Experiments on ε-Samples and Applications

In this section we explore the efficacy of our ε-sample algorithms based on partitions. We use as X the Chicago crime data [1] with roughly 6.5 million data points.

A key step of the analysis is measuring the accuracy of the ε-sample. That is for a sampled S we measure $\mathsf{Error}(X, Y) = \max_{h \in \mathcal{H}_d} |\frac{|Y \cap h|}{|Y|} - \frac{|X \cap h|}{|X|}|$, which unfortunately requires $|X|^{d+1}$ time to simply enumerate, which would be infeasible for large X. Instead we use techniques [18, 2, 17] which provide guaranteed approximation of this function, designed with spatial anomaly detection in mind. We have set the parameters large enough so the noise in computing Error is insignificant compared to the quantities we are evaluating. We evaluated the accuracy and efficiency of computing ε-samples with 8 different methods.

- There are 3 algorithms based on sampling one element per cell from Matousek's partition algorithm with polygonal cells, using tests created by lines Mat Poly Lines, points Mat Poly Points, or the dual approach Mat Poly Dual.
- We consider 2 algorithms based on Chan's partition algorithm Chan and Chan Simple. Each uses polygonal cells and the dual approach for the test set since this specific type of test set allowed for an optimization in the reweighting step. The Chan variant includes subsampling among cells for purpose of reweighting, while the Chan Simple simply uses all of these cells and does not require the sampling step which in practice was inefficient.
- Then Ham Tree Sample draws samples from the cells of the Willards partitioning; these are also cells of a partition, but with worse theoretical size-accuracy bounds.
- Finally we consider two baselines: random sampling, Random Sample, and another approach Biased-L2 [2] which is a greedy, but slow algorithm which has similar worst case guarantees to random sampling, but achieves better error in practice.

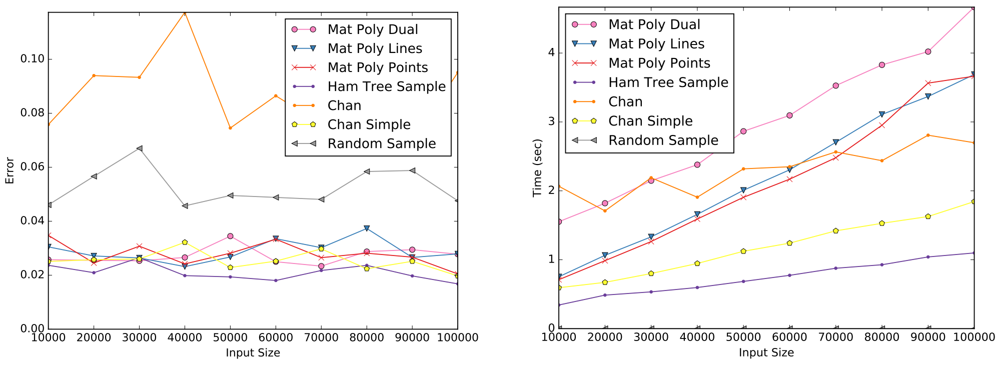


Figure 4 Input size vs. Time and Error using the default parameters.

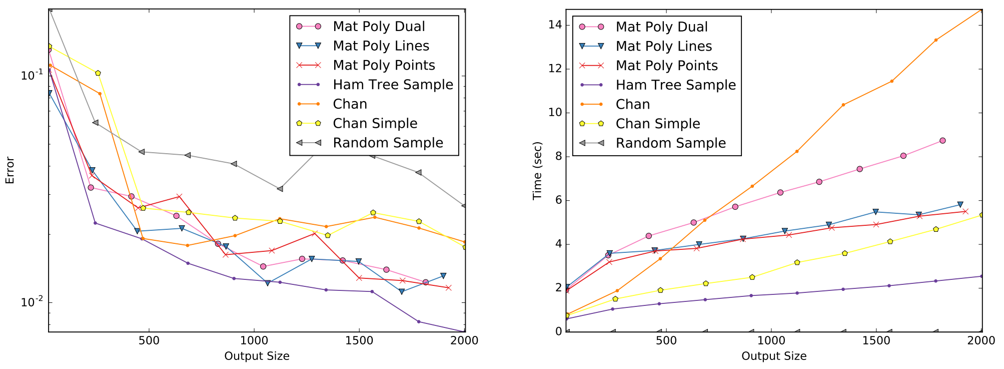


Figure 5 Output size vs. Time and Error using the default parameters.

In testing these algorithms we can control three parameters: the Branching Factor b, the Input Size n (default $n = 100{,}000$ sampled from the crime data set), and the Output Size k (default $k = 1{,}000$). We do not create a sample before creating the partition as analyzed in Theorem 1; we just create the partition on the n points, then sample a point from each cell for the ε-sample. The branching factor only effects ComputePartition-Mat (default $b = 16$) and ComputePartition-Chan (default $b = 22$) and is constant for the execution of the algorithm.

Sample Evaluation Results. We do not plot Biased-L2 since it was quite slow as a function of the Output Size. For $k = 51$ it required 360 seconds which was already more than a factor $100\times$ slower than any other algorithm, and became nearly intractable for $k > 100$. We do note however that its measured Error on small k is competitive with the best of our partitioning based methods.

Figure 3 shows how Branching Factor b affects the time and error. Matoušek-based algorithms seem to gradually decrease in Error, but the trend is very small. For Chan Simple, the Error encounters a phase shift at around $b = 25$, where the error suddenly becomes significantly worse for larger b, probably as an effect of the data set size. The timing is fairly unaffected by b for Chan-based algorithms, but increases noticeably and linearly for the Matoušek based algorithms. We conclude that $b = 22$ is a good choice for Chan-based algorithms and $b = 16$ is a good choice for Matoušek based algorithms.

Figure 4 shows the Input Size relationship to time and Error. As prescribed by the theory, Input Size has no noticeable effect on Error. Moreover, also as expected **the runtime of all algorithms scale linearly with Input Size**.

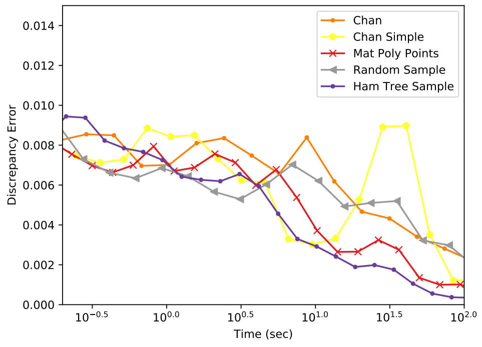


Figure 6 Smooth Discrepancy Error vs. time.

Figure 5 plots the Output Size against the time and Error. As Output Size increases, as expected the error for all methods decreases, note that Error is plotted on a log-scale. The Ham Tree Sample and Mat Poly Lines achieve the smallest Error, with Ham Tree Sample doing the best, and all proposed methods appear to improve upon the old default Random Sample in terms of Error. In particular with Output Size $k = 1000$ both Mat Poly Points and Ham Tree Sample have Error ≈ 0.01 while Random Sample has Error ≈ 0.04. For the Matoušek-based partitioning algorithms, the choice of test set does not have much effect on Error, and perform slightly worse than those based on Chan's partitioning.

Moreover, as Output Size increases the observed run time of all algorithms increases at most linearly. In some cases (e.g., Ham Tree Sample and Mat Poly Lines) the increase is sublinear as these are hierarchical methods, and the largest cost is incurred at the top of the hierarchy. Here as in other plots, we observe that Random Sample is absurdly faster than any other approach. However, even for Output Size $k = 1000$, our methods Ham Tree Sample, Chan Simple, and Mat Poly Points take only about 1, 2.5, and 4 seconds, respectively.

Spatial Anomaly Detection Evaluation. As a concrete demonstration of the usefulness of efficient ε-samples in practice, we apply our new algorithms to a framework for approximately detecting spatial anomalies – maximizing the spatial scan statistic [14]. Specifically each point is endowed with two measures ($b(x)$ the baseline quantity like population and $m(x)$ the measured quantity like disease instance), and let $m(h)$ and $b(h)$ be the fraction of all measured and baseline counts within range $h \in \mathcal{H}_d$, respectively. The main computational problem of exact scan statistics is to find $h^* = \arg\max_{h \in \mathcal{H}_d} \Phi(h)$ where for simplicity we use $\Phi(h) = \phi(m(h), b(h)) = |m(h) - b(h)|$. Approximate scan statistics [18, 17] depend on creating two samples an ε-net which approximates the density of the regions and an ε-samples which approximates the density of points. Together this allows the algorithm to find a $\hat{h}$ where $|\Phi(\hat{h}) - \Phi(h^*)| \leq \varepsilon$; and this is still statistically powerful [18]. In particular, we consider an algorithm for $\hat{h}$ which runs in time $O(n + \frac{1}{\varepsilon}k \log \frac{1}{\varepsilon} + T(n, k))$, where k is the ε-sample size and $T(n, k)$ its construction time. We fix ε to be approximately .0025 which corresponds approximately to an ε-net of size 400. and vary only k. We find approximate anomalies on the crime data set with a particular $h' \in \mathcal{H}_d$ chosen and points chosen, so that $\Phi(h')$ will be anomalously large. Namely we plant a region containing .02 fraction of the points, where in that region points are in the measured set with probability of .7 and baseline set of .3 and outside with probability .5 and .5 respectively. In Figure 6 we plot Discrepancy Error $= |\Phi(\hat{h}) - \Phi(h')|$ as a function of the overall runtime of the algorithms. Note that $\Phi(h^*) \geq \Phi(h')$, so it is possible to find a $\Phi(\hat{h}) \geq \Phi(h')$, but $\Phi(h')$ serves as a useful proxy. We find that Ham Tree Sample generally outperforms Random Sample; for instance for 0.003 error, Ham Tree Sample takes 10 seconds to Random Sample's 50 seconds. Mat Poly Points also usually performs better than Ham Tree Sample, while Chan and Chan Simple perform comparably to random sampling, albeit with high variance, even though their sampling procedure is hundreds of times slower.

Conclusion. Overall we recommend Ham Tree Sample for computing ε-samples if moderate computing beyond random sampling can be tolerated. This method significantly reduces the size and error versus random sampling, and is not difficult to implement.

References

1 Crimes in Chicago. `https://www.kaggle.com/currie32/crimes-in-chicago`, 2017.

2 Huseyin Akcan, Herve Bronnimann, and Robert Marini. Practical and efficient geometric ϵ-approximations. *Proceedings of the 18th Canadian Conference on Computational Geometry*, pages 120–125, 2006.

3 J. Ralph Alexander. Geometric methods in thge theory of uniform distribution. *Combinatorica*, 10:115–136, 1990.

4 Amitabha Bagchi, Amitabh Chaudhary, David Eppstein, and Michael T. Goodrich. Deterministic sampling and range counting in geometric data streams. *ACM Transactions on Algorithms*, 3(A16), 2007.

5 Nikhil Bansal. Constructive algorithms for discrepancy minimization. In *Proceedings 51st Annual IEEE Symposium on Foundations of Computer Science*, pages 407–414, 2010.

6 Christopher Barker. Pep 485 – a function for testing approximate equality. `https://www.python.org/dev/peps/pep-0485/`, Jan 2015.

7 Timothy M. Chan. Optimal partition trees. In *In: Proc. 26th Annu. ACM Sympos. Comput. Geom*, pages 1–10, 2010.

8 Bernard Chazelle. *The Discrepancy Method.* Cambridge, 2000.

9 Bernard Chazelle and Joel Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10:229–249, 1990.

10 Bernard Chazelle and Jiri Matousek. On linear-time deterministic algorithms for optimization problems in fixed dimensions. *Journal of Algorithms*, 21:579–597, 1996.

11 Herbert Edelsbrunner and Emo Welzl. Halfplanar range search in linear space and $o(n^{0.695})$ query time. In 23, editor, *Information Processing Letters*, pages 289–293, 1986.

12 Pavlos S. Efraimidis and Paul G. Spirakis. Weighted random sampling with a reservoir. *Information Processing Letters*, 97(5):181–185, 2006.

13 S. Har-Peled. Constructing planar cuttings in theory and practice. *SIAM J. Comput.*, 29(6):2016–2039, 2000.

14 Martin Kulldorff. A spatial scan statistic. *Communications in Statistics: Theory and Methods*, 26:1481–1496, 1997.

15 Yi Li, Philip M. Long, and Aravind Srinivasan. Improved bounds on the samples complexity of learning. *J. Comp. and Sys. Sci.*, 62:516–527, 2001.

16 Sachar Lovett and Raghu Meka. Constructive discrepancy minimization by walking on the edges. *SIAM Journal on Computing*, 44:1573–1582, 2015.

17 Michael Matheny and Jeff M. Phillips. Computing approximate statistical discrepancy. *CoRR*, abs/1804.11287, 2018. `arXiv:1804.11287`.

18 Michael Matheny, Raghvendra Singh, Liang Zhang, Kaiqiang Wang, and Jeff M. Phillips. Scalable spatial scan statistics through sampling. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2016.

19 Jiri Matoušek. Approximations and optimal geometric divide-and-conquer. In *Proceedings 23rd Symposium on Theory of Computing*, pages 505–511, 1991.

20 Jiri Matoušek. Efficient partition trees. *Discrete & Computational Geometry*, 8:315–334, 1992.

21 Jiri Matoušek. Tight upper bounds for the discrepancy of halfspaces. *Discrete and Computational Geometry*, 13:593–601, 1995.

22 Jiri Matoušek. *Geometric Discrepancy.* Springer, 2009.

23 Jiří Matoušek, Chi-Yuan Lo, and William Steiger. Ham-sandwich cuts in rd. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 539–545, New York, NY, USA, 1992. ACM.

24 Nimrod Megiddo. Partitioning with two lines in the plane. *Journal of Algorithms*, 6(3):430–433, 1985.

25 Raimund Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry*, 1:51–64, 1991.

26 Subhash Suri, Csaba D. Tóth, and Yunhong Zhou. Range counting over multidimensional data streams. In *Proceedings 20th Symposium on Computational Geometry*, pages 160–169, 2004.

27 Vladimir Vapnik and Alexey Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theo. of Prob and App*, 16:264–280, 1971.

28 Ron Wein, Eric Berberich, Efi Fogel, Dan Halperin, Michael Hemmer, Oren Salzman, and Baruch Zukerman. 2D arrangements. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.12 edition, 2018.

29 D. E. Willard. Polygon retrieval. In 11, editor, *SIAM Journal of Computing*, pages 149–165, 1982.

Nearly-Optimal Mergesorts: Fast, Practical Sorting Methods That Optimally Adapt to Existing Runs

J. Ian Munro
University of Waterloo, Canada
imunro@uwaterloo.ca
https://orcid.org/0000-0002-7165-7988

Sebastian Wild
University of Waterloo, Canada
wild@uwaterloo.ca
https://orcid.org/0000-0002-6061-9177

Abstract

We present two stable mergesort variants, “peeksort” and “powersort”, that exploit existing runs and find nearly-optimal merging orders with negligible overhead. Previous methods either require substantial effort for determining the merging order (Takaoka 2009; Barbay & Navarro 2013) or do not have an optimal worst-case guarantee (Peters 2002; Auger, Nicaud & Pivoteau 2015; Buss & Knop 2018). We demonstrate that our methods are competitive in terms of running time with state-of-the-art implementations of stable sorting methods.

2012 ACM Subject Classification Theory of computation → Sorting and searching

Keywords and phrases adaptive sorting, nearly-optimal binary search trees, Timsort

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.63

Related Version `arXiv:1805.04154` (extended version with appendices)

Supplement Material `zenodo:1241162` (code to reproduce running time study)

Funding This work was supported by the Natural Sciences and Engineering Research Council of Canada and the Canada Research Chairs Programme.

1 Introduction

Sorting is a fundamental building block for numerous tasks and ubiquitous in both the theory and practice of computing. While practical and theoretically (close-to) optimal comparison-based sorting methods are known, *instance-optimal sorting,* i.e., methods that *adapt* to the actual input and exploit specific structural properties if present, is still an area of active research. We survey some recent developments in Section 1.1.

Many different structural properties have been investigated in theory. Two of them have also found wide adoption in practice, e.g., in Oracle’s Java runtime library: adapting to the presence of duplicate keys and using existing sorted segments, called *runs*. The former is achieved by a so-called fat-pivot partitioning variant of quicksort [8], which is also used in the OpenBSD implementation of `qsort` from the C standard library. It is an *unstable* sorting method, though, i.e., the relative order of elements with equal keys might be destroyed in the process. It is hence used in Java solely for primitive-type arrays.

© J. Ian Munro and Sebastian Wild;
licensed under Creative Commons License CC-BY

Making use of existing runs in the input is a well-known option in mergesort; e.g., Knuth [17] discusses a bottom-up mergesort variant that does this. He calls it "natural mergesort" and we will use this as an umbrella term for any mergesort variant that picks up existing runs in the input (instead of blindly starting with runs of size 1). The Java library uses *Timsort* [25, 15], a natural mergesort originally developed as Python's new library sort.

While fat-pivot quicksort provably adapts to the *entropy of the multiplicities* of keys [34] – it is optimal up to a factor of 1.088 on average with pseudomedian-of-9 ("ninther") pivots[1] – Timsort is much more heuristic in nature. It picks up existing runs and tries to perform merges in a favorable order (i.e., avoiding merges of runs with very different lengths) by distinguishing a handful of cases of how the lengths of the 4 most recent runs relate to each other. The rules are easy to implement and were empirically shown to be effective in most cases, but their interplay is quite intricate. Although announced as an $O(n \log n)$ worst-case method with its introduction in 2002 [24], a rigorous proof of this bound was only given in 2015 by Auger, Nicaud, and Pivoteau [2] and required a rather sophisticated amortization argument.

The core complication is that – unlike for standard mergesort variants – a given element might participate in more than a logarithmic number of merges. Indeed, Buss and Knop [9] have very recently shown that for some family of inputs, the average number of merges a single element participates in is at least $\bigl(\frac{3}{2} - o(1)\bigr) \cdot \lg n$. So in the worst case, Timsort does, e.g., asymptotically at least *1.5 times as many element moves as standard mergesort.* In terms of adapting to existing order, provable guarantees for Timsort had long remained elusive; an upper bound of $O(n + n \log r)$ was conjectured in [2] and [9]), but indeed only for this very conference, Auger, Jugé, Nicaud and Pivoteau [1] finally give a proof. The hidden constants in their bound are quite big (and far away from the coefficient $\frac{3}{2}$ of the above lower bound).

A further manifestation of the complexity of Timsort's rules was reported by de Gouw et al. [10]: The original rules to maintain the desired invariant for run lengths on the stack are insufficient in some cases. This (algorithmic!) bug had remained unnoticed until their attempt to formally verify the correctness of the Java implementation failed because of it. Gouw et al. proposed two options for correcting Timsort, one of which was applied for the Java library. But now, Auger et al. [1] demonstrated that *this correction is still insufficient:* as of this writing, the Java runtime library contains a flawed sorting method! All of this indicates that already the core algorithm in Timsort is (too?) complicated, and it raises the question whether Timsort's good properties cannot be achieved in a simpler way.

For its theoretical guarantees on adapting to existing runs this is certainly the case. Takaoka [29, 30] and Barbay and Navarro [5] independently described a sorting method that we call *Huffman-Merge* (see below why). It adapts even to the *entropy of the distribution of run lengths:* it sorts an input of r runs with respective lengths $L_1, \ldots, L_r$ in time $O\bigl((\mathcal{H}(\frac{L_1}{n}, \ldots, \frac{L_r}{n}) + 1)n\bigr) \subseteq O(n + n \lg r)$, where $\mathcal{H}(p_1, \ldots, p_r) = \sum_{i=1}^{r} p_i \lg(1/p_i)$ is the binary Shannon entropy.[2] Since $\mathcal{H}(\frac{L_1}{n}, \ldots, \frac{L_r}{n})n - O(n)$ comparisons are necessary for distinct keys, Huffman-Merge is optimal up to $O(n)$. The algorithm is also conceptually simple: find runs in a linear scan, determine an optimal merging order using a Huffman tree of

[1] The median of three elements is chosen as the pivot, each of which is a median of three other elements. This is a good approximation of the median of 9 elements and a recommended pivot selection rule [8].

[2] Note that $\mathcal{H}(L_1/n, \ldots, L_r/n)$ can be significantly smaller than $\lg r$: Consider the run lengths $L_1 = n - \lceil n/\lg n \rceil$ and $L_2 = \cdots = L_r = 1$, i.e., $r = 1 + \lceil n/\lg n \rceil$. Then $\mathcal{H} \le 2$, but $\lg r \sim \lg n$. (Indeed, $\mathcal{H} \to 1$ and the input can indeed be sorted with overall costs $2n$.)

the run lengths, and execute those merges bottom-up in the tree. However, straight-forward implementations add significant overhead in terms of time and space, which renders Huffman-Merge uncompetitive to (reasonable implementations of) elementary sorting methods.

Moreover, Huffman-Merge leads to an *unstable* sorting method since it merges non-adjacent runs. The main motivation for Timsort was to find a fast general-purpose sorting method that is *stable* [24], and the Java library even dictates the sorting method used for objects to be stable.[3] It is conceptually easy to modify the idea of Huffman-Merge to sort stably: replace the Huffman tree by an *optimal binary search tree* and otherwise proceed as before. Since we only have weights at the leaves of the tree, we can compute this tree in $O(n + r \log r)$ time using the Hu-Tucker- or Garsia-Wachs-algorithm. (Barbay and Navarro made this observation, as well; indeed they initially used the Hu-Tucker algorithm [4] and only switched to Huffman in the journal paper [5].) Since r can be $\Theta(n)$ and the algorithms are fairly sophisticated, this idea is not very appealing for use in practical sorting, though.

In this paper, we present two new stable, natural mergesort variants, "peeksort" and "powersort", that have the same optimal asymptotic running time $O\big((\mathcal{H}(\frac{L_1}{n},\ldots,\frac{L_r}{n})+1)n\big)$ as Huffman-merge, but incur much less overhead. For that, we build upon classic algorithms for computing *nearly-optimal binary search trees* [21]; but the vital twist for practical methods is to neither explicitly store the full tree, nor the lengths of all runs at any point in time. In particular – much like Timsort – we only store a *logarithmic* number of runs at any point in time (in fact reducing their number from roughly $\log_\varphi \approx 1.44 \lg n$ in Timsort to $\lg n$), but – much *un*like Timsort – we retain the guarantee of an optimal merging order up to linear terms. Our methods require at most $n \lg n + O(n)$ comparison in the worst case and $\mathcal{H}(\frac{L_1}{n},\ldots,\frac{L_r}{n})n + 3n$ for an input with runs of lengths $L_1,\ldots,L_r$.

We demonstrate in a running-time study that our methods achieve guaranteed (leading-term) optimal adaptive sorting in practice with negligible overhead to compute the merging order: our methods are *not* slower than standard mergesort when no existing runs can be exploited, but outperform standard mergesort and quicksort when long runs are present in the input. Finally, we show that Timsort is slower than standard mergesort and our new methods on certain inputs that do have existing runs, but whose lengths pattern hits a weak point of Timsort's heuristic merging-order rule.

Outline: The rest of this paper is organized as follows. In the remainder of this section we survey related work. Section 2 contains notation and known results on optimal binary search trees that our work builds on. The new algorithms and their analytical guarantees are presented in Section 3. Section 4 reports on our running-time study, comparing the new methods to state-of-the-art sorting methods. Finally, Section 5 summarizes our findings. Details about the experimental setup and some proofs are found in the extended version of this article (`arXiv: 1805.04154`).

1.1 Adaptive Sorting

The idea to exploit existing "structure" in the input to speed up sorting dates (at least) back to methods from the 1970s [20] that sort faster when the number of inversions is small. A systematic treatment of this and many further measures of presortedness (e.g., the number of

[3] We remark that while stability is a much desired feature, practical, stable sorting methods do not try to *exploit* the presence of duplicate elements to speed up sorting, and we will focus on the performance for distinct keys in this article.

inversions, the number of runs, and the number of shuffled up-sequences), their relation and how to sort *adaptively* w.r.t. these measures are discussed by Estivill-Castro and Wood [12].

While the focus of earlier works is mostly on combinatorial properties of permutations, a recent trend is to consider more fine-grained statistical quantities. For example as mentioned above, Huffman-Merge adapts to the *entropy* of the vector of run lengths [29, 30, 5]. Similar measures are the entropy of the lengths of shuffled up-sequences [5] and the entropy of lengths of an LRM-partition [3], a novel measure that lies between runs and shuffled up-sequences.

For multiset sorting, the fine-grained measure, the *entropy of the multiplicities*, has been considered instead of the *number* of unique values already in early work in the field (e.g. [22, 26]). A more recent endeavor has been to find sorting methods that optimally adapt to *both presortedness and repeated values*. Barbay, Ochoa, and Satti refer to this as *synergistic sorting* [6] and present an algorithm based on quicksort that is optimal up to a constant factor. The method's practical performance is unclear.

We remark that (unstable) multiset sorting is the *only* problem from the above list for which a theoretically optimal algorithm has found wide-spread adoption in programming libraries: quicksort is known to almost optimally adapt to the entropy of multiplicities on average [32, 28, 34], when elements equal to the pivot are excluded from recursive calls (fat-pivot partitioning). Supposedly, sorting is so fast to start with that further improvements from exploiting specific input characteristics are only fruitful if they can be realized with minimal additional overhead. Indeed, for algorithms that adapt to the number of inversions, Elmasry and Hammad [11] found that the adaptive methods could only compete with good implementations of elementary sorting algorithms in terms of running time for inputs with extremely few inversions (fewer than 1.5%). Translating the theoretical guarantees of adaptive sorting into practical, efficient methods is an ongoing challenge.

1.2 Lower bound

How much does it help for sorting an array $A[1..n]$ to know that it contains r runs of respective sizes $L_1, \ldots, L_r$, i.e., to know the relative order of $A[1..L_1]$, $A[L_1+1..L_1+L_2]$ etc.? If we assume distinct elements, there are $\binom{n}{L_1,\ldots,L_r}$ permutations that are compatible with this setup, namely the number of ways to partition n keys into r subsets of given sizes. We thus need $\lg(n!) - \sum_{i=1}^{r} \lg(L_i!) = \mathcal{H}(\frac{L_1}{n}, \ldots, \frac{L_r}{n})n - O(n)$ comparisons to sort such an input. A formal argument for this lower bound is given by Barbay and Navarro [5] in the proof of their Theorem 2.

1.3 Results on Timsort and stack-based mergesort

Timsort's good performance in running-time studies, especially on partially sorted inputs, have lead to its adoption in several programming libraries, but until recently, no (nontrivial) worst-case adaptivity guarantee had been known. To make progress towards these, simplified variations of Timsort were considered [2, 9]. They maintain a stack of runs yet to be merged and proceed as follows: Find the next run in the input and push it onto the stack. Then consider the top k elements on the stack (for k a small constant like 3 or 4) and decide based on these if any pair of them is to be merged. If so, the two runs in the stack are replaced with the merged result and the rule is applied repeatedly until the stack satisfies some invariant. The invariant is chosen so as to keep the height of the stack small (logarithmic in n).

The simplest version, "α-stack sort" [2], merges the topmost two runs until the run lengths in the stack grow at least by a factor of α, (e.g., $\alpha = 2$). This method can lead to imbalanced merges (and hence runtime $\omega(n \log r)$ [9]; the authors of [2] also point this out in their conclusion): if the topmost run is much longer than what is below it on the stack, merging the second and third runs (repeatedly until they are at least as big as the topmost run) is much better. This modification is called "α-merge sort". It achieves a worst-case guarantee of $O(n + n\log r)$, but the constant is provably not optimal [9].

Timsort is quite similar to α-merge sort for $\alpha = \varphi$ (the golden ratio) by forcing the run lengths to grow at least like Fibonacci numbers. The detailed rules for selecting merges are found in [1] or [9]. They imply a logarithmic stack height, but the actual bounds are much more involved than it appears at first sight [1]. As mentioned in the introduction, this has lead to the widespread deployment of faulty library code – twice! For inputs with specific run-lengths patterns, the implementations access stack cells beyond the (fixed) stack size.

Timsort was conjectured to always sort in $O(n + n\log r)$ time and in their contribution to these proceedings, Auger et al. [1] finally gave a proof for this bound. The constant of proportionality is not known exactly, but Buss and Knop [9] gave a family of inputs for which Timsort incurs asymptotically at least 1.5 times the required effort (in terms of merge costs, see Section 2.2). This proves that Timsort – like α-merge sort – is *not* optimally adaptive to the *number* of runs r (let alone the entropy of the run length distribution).

2 Preliminaries

Let $A[1..n]$ be an array of n elements that we sort using comparisons. By $\mathcal{H}$, we denote the binary Shannon entropy, i.e., for $p_1, \ldots, p_m \in [0,1]$ with $p_1 + \cdots + p_m = 1$ we have $\mathcal{H}(p_1, \ldots, p_m) = \sum p_i \lg(1/p_i)$, where $\lg = \log_2$. We always let r denote the *number of runs* in the input and $L_1, \ldots, L_r$ their respective lengths with $L_1 + \cdots + L_r = n$. In the literature, a *run* usually means a maximal (contiguous) weakly increasing (i.e., nondecreasing) region, but we adopt the convention from Timsort in this paper: a run is either a maximal *weakly increasing* region *or* a maximal *strictly decreasing* region. Decreasing runs are reversed upon detection; allowing only strictly decreasing runs makes their *stable* reversal trivial. (Note that the algorithms are not affected by these details of what constitutes a "run"; they only rely on a unique partition of the input into sorted segments that can be found by sequential scans.)

2.1 Nearly-Optimal Binary Search Trees

In the *optimal binary search tree problem,* we are given probabilities $\beta_1, \ldots, \beta_m$ to access the m keys $K_1 < \cdots < K_m$ (internal nodes) and probabilities $\alpha_0, \ldots, \alpha_m$ to access the gaps (leaves) between these keys (setting $K_0 = -\infty$ and $K_{m+1} = +\infty$) and we are interested in the binary search tree that minimizes the expected search cost C, i.e., the expected number of (ternary) comparisons when accesses follow the given distribution.[4] Nagaraj [23] surveys various versions of the problem. We confine ourselves to *approximation algorithms* here. Moreover, we only need the special case of *alphabetic trees* where all $\beta_j = 0$.

The following methods apply to the general problem, but we present them for the case of *nearly-optimal alphabetic trees.* So let $\alpha_0, \ldots, \alpha_m$ with $\sum_{i=0}^{m} \alpha_i = 1$ be given. A greedy top-down approach produces provably good search trees if the details are done right [7, 19]: *choose the boundary closest to $\frac{1}{2}$ as the bisection at the root* (*"weight-balancing heuristic"*). Mehlhorn [21, §III.4.2] discusses two algorithms for nearly-optimal binary search trees that follow this scheme: "Method 1" is the straight-forward recursive application of the above rule, whereas "Method 2" (*"bisection heuristic"*) continues by strictly halving the *original* interval in the recursive calls; see Figure 1.

[4] We deviate from the literature convention and use m to denote the number of keys to avoid confusion with n, the length of the arrays to sort, in the rest of the paper.

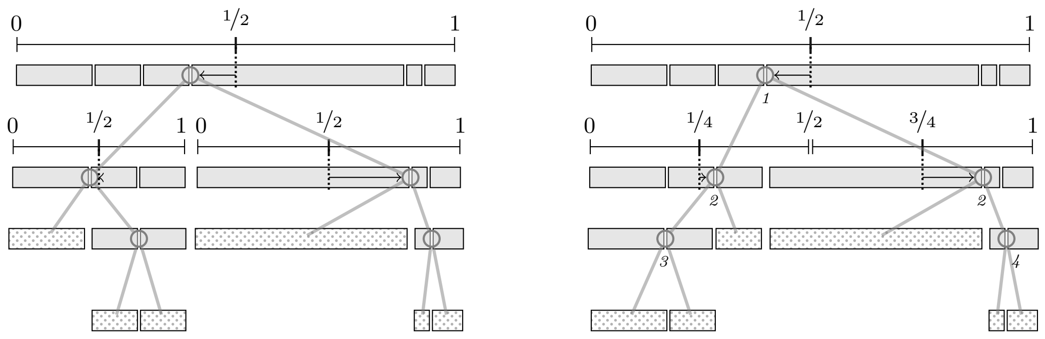


Figure 1 The two versions of weight-balancing for computing nearly-optimal alphabetic trees. The gap probabilities in the example are proportional to $5, 3, 3, 14, 1, 2$. **Left:** Mehlhorn's "Method 1" chooses the split closest to the midpoint of the subtree's actual weights ($1/2$ after renormalization). **Right:** "Method 2" continues to cut the original interval in half, irrespective of the total weight of the subtrees. The italic numbers are the powers of the nodes (see Definition 3 on page 8).

Method 1 was proposed in [31] and analyzed in [18, 7]; Method 2 is discussed in [19]. While Method 1 is arguably more natural, Method 2 has the advantage to yield splits that are predictable without going through all steps of the recursion. Both methods can be implemented to run in time $O(m)$ and yield very good trees. (Recall that in the case $\beta_j = 0$ the classic information-theoretic argument dictates $C \geq \mathcal{H}$; Bayer [7] gives lower bounds in the general case.)

▶ **Theorem 1** (Nearly-Optimal BSTs). *Let* $\alpha_0, \beta_1, \alpha_1, \ldots, \beta_m, \alpha_m \in [0,1]$ *with* $\sum \alpha_i + \sum \beta_j = 1$ *be given and let* $\mathcal{H} = \sum_{i=0}^{m} \alpha_i \lg(1/\alpha_i) + \sum_{j=1}^{m} \beta_j \lg(1/\beta_j)$.

- **(i)** *Method 1 yields a tree with search cost* $C \leq \mathcal{H} + 2$. *[7, Thm 4.8]*
- **(ii)** *If all* $\beta_j = 0$, *Method 1 yields a tree with search cost* $C \leq \mathcal{H} + 2 - (m+3)\alpha_{\min}$, *where* $\alpha_{\min} = \min\{\alpha_0, \ldots, \alpha_m\}$. *[16]*
- **(iii)** *Method 2 yields a tree with search cost* $C \leq \mathcal{H} + 1 + \sum \alpha_i$. *[19]*

2.2 Merge Costs

In this paper, we are primarily concerned with finding a good *order* of binary merges for the existing runs in the input. Following [2] and [9], we define the *merge cost* M for merging two runs of lengths m resp. n as $M = m + n$, i.e., the size of the result. This quantity has been studied earlier by Golin and Sedgewick [14] without giving it a name. Merge costs abstract away from key comparisons and element moves and simplify computations (see next subsection). Since any merge has to move most elements (except for rare lucky cases), and the average number of comparisons using standard merge routines is $m + n - \left(\frac{m}{n+1} + \frac{n}{m+1}\right)$, merge costs are a reasonable approximation, in particular when m and n are roughly equal. They always yield an upper bound for both the number of comparisons and moves.

2.3 Merge Trees

Let $L_1, \ldots, L_r$ with $\sum L_i = n$ be the lengths of the runs in the input. Any natural mergesort can be described as a rule to select some of the remaining runs, which are then merged and replaced by the merge result. If we always merge *two* runs this corresponds to a binary tree with the original runs at leaves $\boxed{1}, \ldots, \boxed{r}$. Internal nodes correspond to the result of merging their children. If we assign to internal node $\textcircled{\scriptsize j}$ the size M_j of the (intermediate) merge result it represents, then the overall merge cost is exactly $M = \sum_{\textcircled{\scriptsize j}} M_j$ (summing over all internal nodes). Figure 1 shows two examples of merge trees; the merge costs are given by adding up all gray areas,[5] (ignoring the dotted leaves).

Let d_i be the *depth* of leaf $\boxed{i}$ (corresponding to the run of length L_i), where depth is the number of edges on the path to the root. Every element in the ith run is counted exactly d_i times in $\sum_{\textcircled{j}} M_j$, so we have $M = \sum_{i=1}^{r} d_i \cdot L_i$. Dividing by n yields $M/n = \sum_{i=1}^{r} d_i \cdot \alpha_i$ for $\alpha_i = L_i/n$, which happens to be the expected search time C in the merge tree if $\boxed{i}$ is requested with probability α_i for $i = 1, \ldots, r$. So the minimal-cost merge tree for given run lengths $L_1, \ldots, L_r$ is the optimal alphabetic tree for leaf probabilities $\frac{L_1}{n}, \ldots, \frac{L_r}{n}$ and it holds $M \ge \mathcal{H}(\frac{L_1}{n}, \ldots, \frac{L_m}{n})n$. For distinct keys, the lower bound on comparisons (Section 1.2) coincides up to linear terms with this lower bound on the merge costs.

Combining this fact with the linear-time methods for nearly-optimal binary search trees from Section 2.1 immediately gives a stable sorting method that adapts optimally to existing runs up to an $O(n)$ term. We call such a method a *nearly-optimal (natural) mergesort*.

3 Nearly-Optimal Merging Orders

We now describe two new nearly-optimal mergesort variants that simulate nearly-optimal search-tree algorithms, but do so without ever storing the entire merge tree or run lengths.

3.1 Peeksort: A Simple Top-Down Method

The first method is similar to standard top-down mergesort in that it implicitly constructs a merge tree on the call stack. Instead of blindly cutting the input in half, however, we mimic Mehlhorn's Method 1. For that we need the *run boundary closest to the middle* of the input: this will become the root of the merge tree. Since we want to detect existing runs anyway at some point, we start by finding the run that contains the middle position. The end point closer to the middle determines the top-level cut and we recursively sort the parts left and right of it. A final merge completes the sort.

To avoid redundant scans, we pass on the information about detected runs. In the general case, we are sorting a range $A[\ell..r]$ whose prefix $A[\ell..e]$ and suffix $A[s..r]$ are (maximal) runs. Depending on whether the middle is contained in one of those runs, we have one of four different cases; apart from that the overall procedure (Algorithm 1) is quite straight-forward.

The following theorem shows that PeekSort is indeed a nearly-optimal mergesort. Unlike previous such methods, its code has very little overhead – in terms of both time and space, as well as in terms of conceptual complexity – over standard top-down mergesort, so it is a promising method for a practical nearly-optimal mergesort.

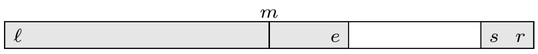
▶ **Theorem 2.** *The merge cost of* PeekSort *on an input consisting of r runs with respective lengths $L_1, \ldots, L_r$ is at most $\mathcal{H}(\frac{L_1}{n}, \ldots, \frac{L_m}{n})n + 2n - (r+2)$, the number of comparisons is at most $\mathcal{H}(\frac{L_1}{n}, \ldots, \frac{L_m}{n}) + 3n - (2r+3)$. Both is optimal up to $O(n)$ terms (in the worst case).*

Proof. The recursive calls of Algorithm 1 produce the same tree as Mehlhorn's Method 1 with input $(\alpha_0, \ldots, \alpha_m) = (\frac{L_1}{n}, \ldots, \frac{L_r}{n})$ (i.e., $m = r - 1$) and $\beta_j = 0$. By Theorem 1–(ii), the search costs in this tree are $C \le \mathcal{H} + 2 - (m+3)\alpha_{\min}$ with $\mathcal{H} = \mathcal{H}(\frac{L_1}{n}, \ldots, \frac{L_m}{n})$. Since $L_j \ge 1$, we have $\alpha_{\min} \ge \frac{1}{n}$. As argued in Section 2.3, the overall merge costs are then given by $M = Cn \le \mathcal{H}n + 2n - (r+2)$, which is within $O(n)$ of the lower bound for M.

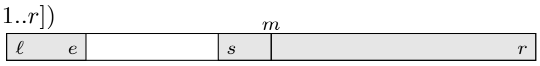
[5] The left tree is obviously better here and this is a typical outcome. But there are also inputs where Method 2 yields a better tree than Method 1.

Algorithm 1 Peeksort: A simple top-down version of nearly-optimal natural mergesort. The initial call is PEEKSORT($A[1..n], 1, n$). Procedures EXTENDRUNLEFT (-RIGHT) scan left (right) starting at $A[m]$ as long as the run continues (and we did not cross the second parameter).

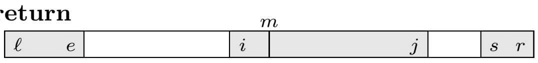
```
PEEKSORT($A[\ell..r], e, s$)
if $e == r$ or $s == \ell$ then return
$m = \ell + \lfloor \frac{r-\ell}{2} \rfloor$
if $m \le e$                //
```



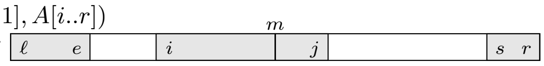
```
    PEEKSORT($A[e+1..r], e+1, s$)
    MERGE($A[\ell..e], A[e+1..r]$)
else if $m \ge s$           //
```



```
    PEEKSORT($A[\ell..s-1], e, s-1$)
    MERGE($A[\ell..s-1], A[s..r]$)
else    // Find existing run $A[i..j]$ containing position $m$
    $i$ = EXTENDRUNLEFT($A[m], \ell$);   $j$ = EXTENDRUNRIGHT($A[m], r$)
    if $i == \ell$ and $j == r$ return
    if $m - i < j - m$      //
```



```
        PEEKSORT($A[\ell..i-1], e, i-1$)
        PEEKSORT($A[i..r], j, s$)
        MERGE($A[\ell..i-1], A[i..r]$)
    else                    //
```



```
        PEEKSORT($A[\ell..j], e, i$)
        PEEKSORT($A[j+1..r], j+1, s$)
        MERGE($A[\ell..j], A[j+1..r]$)
```

We can save at least one comparison per merge (namely for the last element). In total, we do exactly $r-1$ merge operations. Apart from merging, we need a total of $n-1$ additional comparisons to detect the existing runs in the input. Barbay and Navarro [5, Thm. 2] argued that $\mathcal{H}n - O(n)$ comparisons are necessary if the elements in the input are all distinct. ◀

3.2 Powersort: A Cache-Friendly Stack-Based Method

One little blemish remains in PEEKSORT: we have to use "random accesses" into the middle of the array to decide how to proceed. Even though we only use cache-friendly sequential scans, the I/O operations to load the middle run are effectively wasted, since it will often be merged only much later (after further recursive calls). Timsort proceeds in one left-to-right scan over the input and merges the top runs on the stack. This increases the likelihood to still have (parts of) the most recently detected run in cache when it is merged subsequently.

3.2.1 The power of top-down in a bottom-up method

In Method 2 for nearly-optimal search trees, we can unfold the recursion since all its potential cut points are predetermined: they have the form $c \cdot 2^{-\ell}$ for $\ell \in \mathbb{N}_{\ge 1}$ and odd c. The following definition characterizes the recursion depth ℓ, at which an internal node $\textcircled{j}$ is considered.

▶ **Definition 3** (Node Power). Let $\alpha_0, \ldots, \alpha_m$, $\sum \alpha_j = 1$ be leaf probabilities. For $1 \le j \le m$, let $\textcircled{j}$ be the internal node separating the $(j-1)$st and jth leaf. The *power* of (the split at)

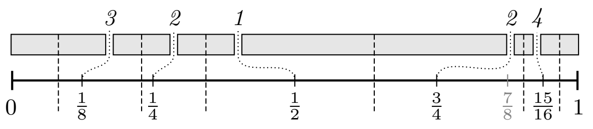


Figure 2 Illustration of node powers. The power of a run boundary is the smallest ℓ so that a multiple of $2^{-\ell}$ lies between the midpoints of the runs it separates.

node $\textcircled{j}$ is

$$P_j = \min\left\{\ell \in \mathbb{N} : \left\lfloor \frac{a}{2^{-\ell}} \right\rfloor < \left\lfloor \frac{b}{2^{-\ell}} \right\rfloor\right\}, \quad \text{where } a = \sum_{i=0}^{j-1} \alpha_i - \tfrac{1}{2}\alpha_{j-1}, \quad b = \sum_{i=0}^{j-1} \alpha_i + \tfrac{1}{2}\alpha_j.$$

(P_j is the index of the first bit where the (binary) fractional parts of a and b differ.)

Figure 2 illustrates Definition 3 for the nodes in our example (Figure 1). Intuitively, P_j is the "intended" depth of $\textcircled{j}$, but nodes occasionally end up higher in the tree if some leaf has a large weight relative to the current subtree, (see the rightmost branch in Figure 1). Mehlhorn's [19, 21] original implementation of Method 2, procedure *construct-tree*, does not single out the case that the next desired cut point lies *outside* the range of a subtree (cf. $\frac{7}{8}$ in Figure 2). This reduces the number of cases, but for our application, it is more convenient to explicitly check for this out-of-range case, and if it occurs to directly proceed to the next cut point. We refer to the modified algorithm as *Method 2′*. The extended version of this paper gives the details and shows that the changes do not affect the guarantee for Method 2 in Theorem 1. With this modification, the resulting tree is characterized by the node powers.

▶ **Lemma 4** (Path power monotonicity). *Consider the tree constructed by Method 2′. The powers of internal nodes along any root-to-leaf path are strictly increasing.*

The proof is given in the extended version.

3.2.2 Merging on-the-fly

Our second algorithm, "powersort", constructs the merge tree from left to right. Whenever the next internal node has a smaller power than the preceding one, we are following an edge from a left child up to its parent. That means that this subtree does not change anymore and we can execute any pending merges in it before continuing; the subtree then corresponds to the single resulting run. If we are instead following an edge down to a right child of a node, that subtree is still "open" and we only push the corresponding run onto the stack.

▶ Remark. The tree constructed by Method 2′ for leaf probabilities $\alpha_0, \ldots, \alpha_m$ is the (unique, min-oriented) Cartesian tree[6] for the sequence of node powers $P_1, \ldots, P_m$. It can be constructed iteratively (left to right) by the algorithm of Gabow, Bentley, and Tarjan [13], which effectively coincides with the procedure sketched above, except that we immediately collapse (merge) completed subtrees instead of keeping the entire tree.

As the runs on the stack correspond to a (right) path in the tree, the nodes have strictly increasing powers and we can bound the stack height by the maximal P_j. Since our leaf probabilities are $\alpha_j = \frac{L_j}{n} \geq \frac{1}{n}$, we have $P_j \leq \lfloor \lg n \rfloor + 1$. Algorithm 2 shows the detailed code.

[6] The Cartesian tree of an array of numbers is a binary tree. Its root corresponds to the global minimum in the array and its subtrees are the Cartesian trees of the numbers left resp. right of the minimum.

Algorithm 2 Powersort: A one-pass stack-based nearly-optimal natural mergesort. Procedure ExtendRunRight scans right as long as the run continues.

```
PowerSort($A[1..n]$)
X = stack of runs                (capacity $\lfloor \lg(n) \rfloor + 1$)
P = stack of powers              (capacity $\lfloor \lg(n) \rfloor + 1$)
$s_1 = 1$;  $e_1 = $ ExtendRunRight($A[1], n$)    // $A[s_1..e_1]$ is leftmost run
while $e_1 < n$
    $s_2 = e_1 + 1$;  $e_2 = $ ExtendRunRight($A[s_2], n$)    // $A[s_2..e_2]$ next run
    $p = $ NodePower($s_1, e_1, s_2, e_2, n$)    // $P_j$ for node $\textcircled{j}$ between $A[s_1..e_1]$ and $A[s_2..e_2]$
    while $P.top() > p$    // previous merge deeper in tree than current
        $P.pop()$          // $\rightsquigarrow$ merge and replace run $A[s_1..e_1]$ by result
        $(s_1, e_1) = $ Merge($X.pop()$, $A[s_1..e_1]$)
    end while
    $X.push(A[s_1, e_1])$;  $P.push(p)$
    $s_1 = s_2$;  $e_1 = e_2$
end while // Now $A[s_1..e_1]$ is the rightmost run
while $\neg X.empty()$
    $(s_1, e_1) = $ Merge($X.pop()$, $A[s_1..e_1]$)
end while
```

```
NodePower($s_1, e_1, s_2, e_2, n$)
$n_1 = e_1 - s_1 + 1$;  $n_2 = e_2 - s_2 + 1$;  $\ell = 0$
$a = (s_1 + n_1/2 - 1)/n$;  $b = (s_2 + n_2/2 - 1)/n$
while $\lfloor a \cdot 2^\ell \rfloor == \lfloor b \cdot 2^\ell \rfloor$ do $\ell = \ell + 1$ end while
return $\ell$
```

▶ **Theorem 5.** *The merge cost of* PowerSort *is at most* $\mathcal{H}(\frac{L_1}{n}, \ldots, \frac{L_m}{n}) + 2n$, *the number of comparisons is at most* $\mathcal{H}(\frac{L_1}{n}, \ldots, \frac{L_m}{n}) + 3n - r$. *Moreover, (apart from buffers for merging) only* $O(\log n)$ *words of extra space are required.*

Proof. The merge tree of PowerSort is exactly the search tree constructed by Method 2′ on leaf probabilities $(\alpha_0, \ldots, \alpha_m) = (\frac{L_1}{n}, \ldots, \frac{L_r}{n})$ and $\beta_j = 0$. By Theorem 1–(iii), the search costs are $C \le \mathcal{H} + 2$ with $\mathcal{H} = \mathcal{H}(\frac{L_1}{n}, \ldots, \frac{L_m}{n})$, so the overall merge costs are $M = Cn \le \mathcal{H}n + 2n$, which is within $O(n)$ of the lower bound for M. The comparisons follow as for Theorem 2. ◀

4 Running-Time Study

We conducted a running-time study comparing the two new nearly-optimal mergesorts with current state-of-the-art implementations and elementary mergesort variants. The code is available on github [33]. The main goal of this study is to show that

1. peeksort and powersort have very little overhead compared to standard (non-natural) mergesort variants (i.e., they are never (much) slower); and at the same time
2. peeksort and powersort outperform other mergesort variants on partially presorted inputs.

Timsort is arguably the most used adaptive sorting method; a secondary goal is hence to

3. investigate the typical merge costs of Timsort on different inputs, in particular in light of the recent negative theoretical results by Buss and Knop [9].

4.1 Setup

Oracle's Java runtime library includes a highly tuned Timsort implementation; to be able to directly compare with it, we chose to implement our algorithms in Java. We repeated all experiments with C++ ports of the algorithms and the results are qualitatively the same; we will comment on a few differences in the following subsections. The Timsort implementation is used for `Object[]`, i.e., arrays of *references* to objects. Since the location of objects on the heap is hard to control, this is likely to yield big variations in the number of cache misses. We chose to sort `int[]`s instead to obtain more reproducible results, and modified the library implementation of Timsort accordingly. This scenario makes key comparisons and element moves relatively cheap and thereby emphasizes the remaining overhead of the methods, which is in line with our primary goal 1) from above.

We compare our methods with standard top-down and bottom-up mergesort implementations. We use the code given by Sedgewick [27, Programs 8.3 and 8.5] as bases. In both cases, we add a check before calling merge to detect if the runs happen to already be in sorted order, and we use insertion sort for base cases of size $n \le w = 24$. (For bottom-up mergesort, we start by sorting chunks of size $w = 24$.) Our implementations of peeksort and powersort are described in more detail in the extended version; apart from a mildly optimized version of the pseudocode, we added the same cutoff / minimal run length ($w = 24$) as above.

All our methods use Sedgewick's bitonic merge procedure [27, Program 8.2], whereas the Java library Timsort contains a custom merge method that tries to save key comparisons: when only elements from the same run are selected for the output repeatedly, Timsort enters a *"galloping mode"* and uses exponential search (instead of the conventional sequential search) to find the insertion position of the next element. Details are described by Peters [25]. Since saving comparisons is not of utmost importance in our scenario of sorting `int`s, we also added a version of Timsort, called "trotsort", that uses our plain merge method instead of galloping, but is otherwise identical to the library Timsort.

We use the following inputs types:

- *random permutations* are a case where no long runs are present to exploit;
- *"random-runs" inputs* are constructed from a random permutation by sorting segments of random lengths, where the lengths are chosen independently according to a geometric distribution with a given mean ℓ; since the geometric distribution has fairly large variance, these inputs tend to have runs whose sizes vary a lot;
- *"Timsort-drag" inputs* are special instances of random-runs inputs where the run lengths are chosen as $\mathcal{R}_{\mathrm{tim}}$, the bad-case example for Timsort from [9, Thm. 3].

We remark that the above input types are chosen with our specific goals from above in mind; we do not attempt to model inputs from "real-world" applications in this study.

4.2 Overhead of Nearly-Optimal Merge Order

We first consider random permutations as inputs. Since random permutations contain (with high probability) no long runs that can be exploited, the adaptive methods will not find anything that would compensate for their additional efforts to identify runs. (This is confirmed by the fact that the total merge costs of all methods, including Timsort, are within 1.5% of each other in this scenario.) Figure 3 shows average running times for input sizes from 100 000 to 100 million ints. (Measurements for $n = 10\,000$ were too noisy to draw meaningful conclusions.)

Varying input sizes over several orders of magnitude, we consistently see the following picture: `Arrays.sort(int[])` (dual-pivot quicksort) is the fastest method. It is not a stable

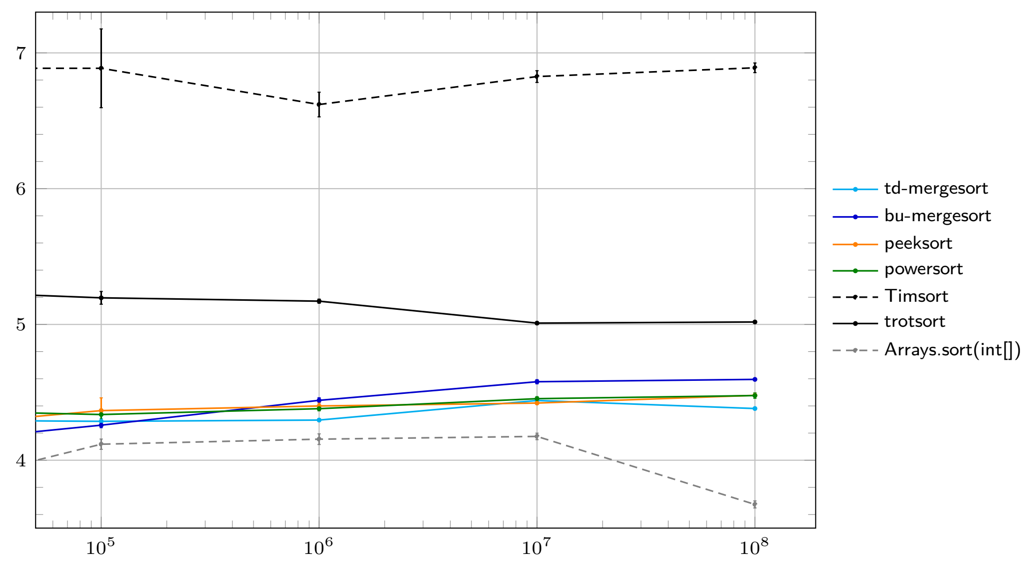


Figure 3 Normalized running times on random permutations. The logarithmic x-axis shows n, the y-axis shows the average of $t/(n \lg n)$ where t is the running time in ms. Error bars show one standard deviation (stdev). We plot averages over 1000 repetitions (200 resp. 20 for the largest n).

sort and thus merely serves as a baseline. Top-down and bottom-up mergesort, peeksort and powersort are 20–30% slower than dual-pivot quicksort. Comparing the four to each other, no large differences are visible; if anything, bottom-up mergesort was a bit slower than the others (for large inputs). Since the recursion cutoff resp. minimal run length $w = 24$ exceeded the length of *all* runs in all inputs, we effectively have equal-length runs. Merging left to right (as in bottom-up mergesort) then performs just fine, and top-down mergesort finds a close-to-optimal merging order in this case. That peek- and powersort perform essentially *as good as* elementary mergesorts on random permutations thus clearly indicates that their overhead for determining a nearly-optimal merging order is negligible.

The library Timsort performs surprisingly poorly on `int[]`s, probably due to the relatively cheap comparisons. Replacing the galloping merge with the ordinary merge alleviates this (see “trotsort”), but Timsort remains inferior on random permutations by a fair margin (10–20%). In C++, trotsort with straight insertion sort (instead of binary insertion sort) as base case was roughly as fast as the other mergesorts.

4.3 Practical speedups by adaptivity

After demonstrating that we do not lose much by using our adaptive methods when there is nothing to adapt to, we next investigate how much can be gained if there is. We consider random-runs inputs as described above. This input model instills a good dose of presortedness, but not in a way that gives any of the tested methods an obvious advantage or disadvantage over the others. We choose a representative size of $n = 10^7$ and an expected run length $\ell = 3\,000$, so that we expect roughly $\sqrt{n}$ runs of length $\sqrt{n}$.

If this was a random permutation, we would expect merge costs of roughly $n \lg(n/w) \approx 1.87 \cdot 10^8$ (indeed a bit above this). The right chart in Figure 4 shows that the adaptive methods can bring the merge cost down to a little over 60% of this number. Powersort achieved average merge costs of $1.14 \cdot 10^8 < n \lg r \approx 1.17 \cdot 10^8$, i.e., less than a method would that only adapts to the *number* of runs r.

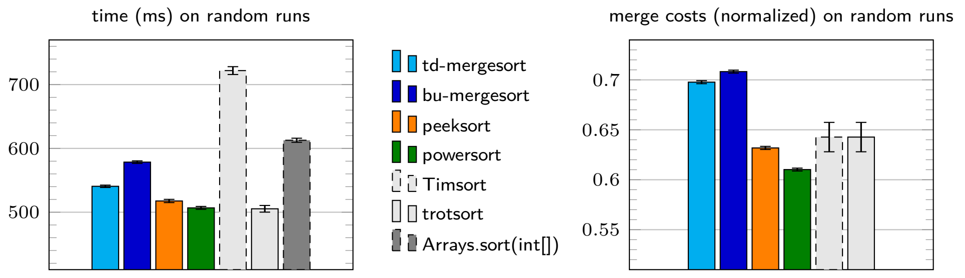


Figure 4 Average running times (left) and normalized merge cost (right) on random-runs inputs with $n = 10^7$ and $\ell = 3\,000 \approx \sqrt{n}$. Error bars show one stdev. Merge costs have been divided by $n \lg(n/w) \approx 1.87 \cdot 10^8$, which is the merge cost a (hypothetical) optimal mergesort that does not pick up existing runs, but starts with runs of length $w = 24$. Note that the simple sorted-check before a merge in the standard mergesorts already reduces the merge costs to roughly 70% of that number.

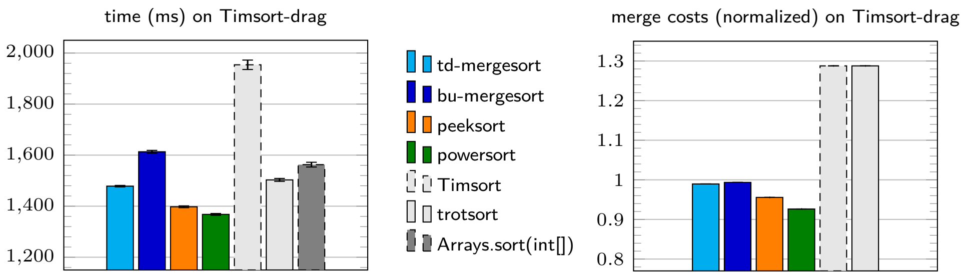


Figure 5 Average running times (left) and normalized merge cost (right) on “Timsort-drag” inputs with $n = 2^{24}$ and run lengths $\mathcal{R}_{\text{tim}}(2^{24}/32)$ multiplied by 32. Error bars show one standard deviation. Merge costs have been divided by $n \lg(n/w) \approx 3.26 \cdot 10^8$.

In terms of running time, powersort is again among the fastest stable methods, and indeed 20% *faster* than `Arrays.sort(int[])`. The best adaptive methods are also 10% faster than top-down mergesort. (Note that our standard mergesorts are “slightly adaptive” by skipping a merge of two runs that happened to already be in order, which can be checked with a single comparisons.) This supports the statement that significant speedups can be realized by adaptive sorting on inputs with existing order, and $\sqrt{n}$ runs suffice for that. If we increase ℓ to 100 000, so that we expect only roughly 100 long runs, the library quicksort becomes twice as slow as powersort and trotsort.

Timsort with galloping merges is again uncompetitive. Trotsort's running time is a bit anomalous. Even though it occasionally incurs 10% more merge costs on a given input than powersort, the Java running times were within 1% of each other. However, merge costs seem to more accurately predict running time in C++; there trotsort was 8% slower than powersort.

4.4 Non-optimality of Timsort

Finally, we consider “Timsort-drag” inputs, a sequence of run lengths $\mathcal{R}_{\text{tim}}(n)$ specifically crafted by Buss and Knop [9] to generate unbalanced merges (and hence large merge cost) in Timsort. Since actual Timsort implementations employ minimal run lengths of up to 32 elements we multiplied the run lengths by 32. Figure 5 shows running time and merge cost for all methods on a characteristic Timsort-drag input of length $2^{24} \approx 1.6 \cdot 10^7$.

In terms of merge costs, Timsort resp. trotsort now pays 30% more than even a simple non-adaptive mergesort, whereas peeksort and powersort obviously retain their proven nearly-optimal behavior. Also in terms of running time, trotsort is a bit slower than top-down mergesort, and 10% slower than powersort on these inputs. It is remarkable that the dramatically larger merge cost does not lead to a similarly drastic slow down in Java. In C++, however, trotsort was *40% slower* than powersort, in perfect accordance with the difference in merge costs.

It must thus be noted that Timsort's merging-order rules have weaknesses, and it is unclear if more dramatic examples are yet to be found.

5 Conclusion

In this paper, we have demonstrated that provably good merging orders for natural mergesort can be found with negligible overhead. The proposed algorithms, peeksort and powersort, offer more reliable performance than the widely used Timsort, and at the same time, are arguably simpler.

Powersort builds on a modified bisection heuristic for computing nearly-optimal binary search trees that might be of independent interest. It has the same quality guarantees as Mehlhorn's original formulation, but allows the tree to be built "bottom-up" as a Cartesian tree over a certain sequence, the "node powers". It is the only such method for nearly-optimal search trees to our knowledge.

Buss and Knop conclude with the question whether there exists a k-aware algorithm (a stack-based natural mergesort that only considers the top k runs in the stack) with merge cost $(1 + o_r(1))$ times the optimal merge cost [9, Question 37]. Powersort effectively answers this question in the affirmative with $k = 3$.[7]

5.1 Extensions and future work

Timsort's "galloping merge" procedure saves comparisons when we consistently consume elements from one run, but in "well-mixed" merges, it does not help (much). It would be interesting to compare this method with other comparison-efficient merge methods.

Another line of future research is to explore ways to profit from duplicate keys in the input. The ultimate goal would be a "synergistic" sorting method (in the terminology of [6]) that has practically no overhead for detecting existing runs and equals and yet exploits their *combined* presence optimally.

References

1. Nicolas Auger, Vincent Jugé, Cyril Nicaud, and Carine Pivoteau. On the worst-case complexity of TimSort. In Hannah Bast Yossi Azar and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, Leibniz International Proceedings in Informatics (LIPIcs), 2018. doi:10.4230/LIPIcs.ESA.2018.4.
2. Nicolas Auger, Cyril Nicaud, and Carine Pivoteau. Merge strategies: from merge sort to TimSort, 2015. URL: https://hal-upec-upem.archives-ouvertes.fr/hal-01212839.

[7] Strictly speaking, powersort needs a relaxation of the model of Buss and Knop. They require decisions to be made solely based on the *lengths* of runs, whereas node power takes the location of the runs within the array into account. Since the location of a run must be stored anyway, this appears reasonable to us.

3 Jérémy Barbay, Johannes Fischer, and Gonzalo Navarro. LRM-trees: Compressed indices, adaptive sorting, and compressed permutations. *Theoretical Computer Science*, 459:26–41, 2012. doi:10.1016/j.tcs.2012.08.010.

4 Jérémy Barbay and Gonzalo Navarro. Compressed representations of permutations, and applications. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009)*, pages 111–122, Freiburg, Germany, 2009. URL: https://hal.inria.fr/inria-00358018.

5 Jérémy Barbay and Gonzalo Navarro. On compressing permutations and adaptive sorting. *Theoretical Computer Science*, 513:109–123, 2013. doi:10.1016/j.tcs.2013.10.019.

6 Jérémy Barbay, Carlos Ochoa, and Srinivasa Rao Satti. Synergistic solutions on multisets. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 31:1–31:14, 2017. doi:10.4230/LIPIcs.CPM.2017.31.

7 Paul J. Bayer. *Improved Bounds on the Cost of Optimal and Balanced Binary Search Trees*. Master's thesis, Massachusetts Institute of Technology, 1975.

8 Jon L. Bentley and M. Douglas McIlroy. Engineering a sort function. *Software: Practice and Experience*, 23(11):1249–1265, 1993. doi:10.1002/spe.4380231105.

9 Sam Buss and Alexander Knop. Strategies for stable merge sorting, 2018. arXiv:1801.04641.

10 Stijn de Gouw, Frank S. de Boer, Richard Bubel, Reiner Hähnle, Jurriaan Rot, and Dominic Steinhöfel. Verifying OpenJDK's sort method for generic collections. *Journal of Automated Reasoning*, aug 2017. doi:10.1007/s10817-017-9426-4.

11 Amr Elmasry and Abdelrahman Hammad. Inversion-sensitive sorting algorithms in practice. *Journal of Experimental Algorithmics*, 13:11:1–11:18, 2009. doi:10.1145/1412228.1455267.

12 Vladmir Estivill-Castro and Derick Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24(4):441–476, dec 1992. doi:10.1145/146370.146381.

13 Harold N. Gabow, Jon Louis Bentley, and Robert E. Tarjan. Scaling and related techniques for geometry problems. In *16th annual ACM symposium on Theory of computing (STOC 1984)*, pages 135–143. ACM Press, 1984. doi:10.1145/800057.808675.

14 Mordecai J. Golin and Robert Sedgewick. Queue-mergesort. *Information Processing Letters*, 48(5):253–259, dec 1993. doi:10.1016/0020-0190(93)90088-q.

15 Chris Hegarty. Replace "modified mergesort" in java.util.Arrays.sort with timsort, 2009. URL: https://bugs.openjdk.java.net/browse/JDK-6804124.

16 Yasuichi Horibe. An improved bound for weight-balanced tree. *Information and Control*, 34(2):148–151, jun 1977. doi:10.1016/S0019-9958(77)80011-9.

17 Donald E. Knuth. *The Art Of Computer Programming: Searching and Sorting*. Addison Wesley, 2nd edition, 1998.

18 Kurt Mehlhorn. Nearly optimal binary search trees. *Acta Informatica*, 5(4), 1975. doi:10.1007/BF00264563.

19 Kurt Mehlhorn. A best possible bound for the weighted path length of binary search trees. *SIAM Journal on Computing*, 6(2):235–239, 1977. doi:10.1137/0206017.

20 Kurt Mehlhorn. Sorting presorted files. In *Theoretical Computer Science 4th GI Conference*, pages 199–212. Springer, 1979. doi:10.1007/3-540-09118-1_22.

21 Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*. Springer, 1984.

22 Ian Munro and Philip M. Spira. Sorting and searching in multisets. *SIAM Journal on Computing*, 5(1):1–8, 1976. doi:10.1137/0205001.

23 S.V. Nagaraj. Optimal binary search trees. *Theoretical Computer Science*, 188(1-2):1–44, nov 1997. doi:10.1016/S0304-3975(96)00320-9.

24 Tim Peters. [Python-Dev] Sorting, 2002. URL: https://mail.python.org/pipermail/python-dev/2002-July/026837.html.

25 Tim Peters. Timsort, 2002. URL: http://hg.python.org/cpython/file/tip/Objects/listsort.txt.

26 Robert Sedgewick. Quicksort with equal keys. *SIAM Journal on Computing*, 6(2):240–267, 1977.

27 Robert Sedgewick. *Algorithms in Java*. Addison-Wesley, 2003.

28 Robert Sedgewick and Jon Bentley. Quicksort is optimal (talk slides), 2002. URL: http://www.cs.princeton.edu/~rs/talks/QuicksortIsOptimal.pdf.

29 Tadao Takaoka. A new measure of disorder in sorting – entropy. In *The Fourth Australasian Theory Symposium (CATS 1998)*, pages 77–86, 1998.

30 Tadao Takaoka. Partial solution and entropy. In *MFCS 2009*, pages 700–711, 2009. doi:10.1007/978-3-642-03816-7_59.

31 W.A. Walker and C.C. Gotlieb. A top-down algorithm for constructing nearly optimal lexicographic trees. In *Graph Theory and Computing*, pages 303–323. Elsevier, 1972. doi:10.1016/B978-1-4832-3187-7.50023-4.

32 Lutz M. Wegner. Quicksort for equal keys. *IEEE Transactions on Computers*, C-34(4):362–367, 1985. doi:10.1109/TC.1985.5009387.

33 Sebastian Wild. Accompanying code for running time study, 2018. doi:10.5281/zenodo.1241162.

34 Sebastian Wild. Quicksort is optimal for many equal keys. In *ANALCO 2018*, pages 8–22. SIAM, jan 2018. doi:10.1137/1.9781611975062.2.

On a Problem of Danzer

Nabil H. Mustafa[1]
Université Paris-Est, Laboratoire d'Informatique Gaspard-Monge, Equipe A3SI, ESIEE Paris
mustafan@esiee.fr

Saurabh Ray
Department of Computer Science, NYU Abu Dhabi, United Arab Emirates
saurabh.ray@nyu.edu

Abstract

Let C be a bounded convex object in $\mathbb{R}^d$, and P a set of n points lying outside C. Further let c_p, c_q be two integers with $1 \leq c_q \leq c_p \leq n - \lfloor \frac{d}{2} \rfloor$, such that every $c_p + \lfloor \frac{d}{2} \rfloor$ points of P contains a subset of size $c_q + \lfloor \frac{d}{2} \rfloor$ whose convex-hull is disjoint from C. Then our main theorem states the existence of a partition of P into a small number of subsets, each of whose convex-hull is disjoint from C. Our proof is constructive and implies that such a partition can be computed in polynomial time.

In particular, our general theorem implies polynomial bounds for Hadwiger-Debrunner (p, q) numbers for balls in $\mathbb{R}^d$. For example, it follows from our theorem that when $p > q \geq (1 + \beta) \cdot \frac{d}{2}$ for $\beta > 0$, then any set of balls satisfying the HD(p, q) property can be hit by $O\left(q^2 p^{1+\frac{1}{\beta}} \log p\right)$ points. This is the first improvement over a nearly 60-year old exponential bound of roughly $O\left(2^d\right)$.

Our results also complement the results obtained in a recent work of Keller *et al.* where, apart from improvements to the bound on HD(p, q) for convex sets in $\mathbb{R}^d$ for various ranges of p and q, a polynomial bound is obtained for regions with low union complexity in the plane.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Convex polytopes, Hadwiger-Debrunner numbers, Epsilon-nets, Balls

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.64

1 Introduction

Given a finite set $\mathcal{C}$ of geometric objects in $\mathbb{R}^d$, we say that $\mathcal{C}$ satisfies the HD(p, q) property if for any set $\mathcal{C}' \subseteq \mathcal{C}$ of size p, there exists a point in $\mathbb{R}^d$ common to at least q objects of $\mathcal{C}'$. The goal then is to show that there exists a small set Q of points in $\mathbb{R}^d$ such that each object of $\mathcal{C}$ contains some point of Q; such a Q is called a hitting set for $\mathcal{C}$.

These bounds for a set $\mathcal{C}$ of convex sets in $\mathbb{R}^d$ have been studied since the 1950s (see the surveys [7, 8, 15]), and it was only in 1991 that Alon and Kleitman [1], in a breakthrough result, gave an upper-bound that is *independent* of $|\mathcal{C}|$. Unfortunately it depends exponentially on p, q and d. For the case where $\mathcal{C}$ consists of arbitrary convex objects, the current best bounds remain exponential in p, q and d.

[1] The work of Nabil H. Mustafa in this paper has been supported by the grant ANR SAGA (JCJC-14-CE25-0016-01).

© Nabil H. Mustafa and Saurabh Ray;
licensed under Creative Commons License CC-BY

▶ **Theorem A** ([1, 9]). *Let $\mathcal{C}$ be a finite set of convex objects in $\mathbb{R}^d$ satisfying the* $\mathrm{HD}(p,q)$ *property, where p, q are two integers with $p \geq q \geq d+1$. Then there exists a hitting set for $\mathcal{C}$ of size*

$$\begin{cases} O\left(p^{d\frac{q-1}{q-d}} \cdot \log^{c' d^3 \log d} p\right), & \\ (p-q) + O\left(\left(\frac{p}{q}\right)^d \log^{c' d^3 \log d}\left(\frac{p}{q}\right)\right), & \text{for } \; q \geq \log p \\ p - q + 2, & \text{for } \; q \geq p^{1-\frac{1}{d}+\epsilon}, p \geq p(d,\epsilon). \end{cases}$$

where c' is an absolute constant independent of $|\mathcal{C}|, p, q$ and d, and $p(d,\epsilon)$ is a function depending only on d and ϵ.

Consider the basic case where $\mathcal{C}$ is a set of balls in $\mathbb{R}^d$ satisfying the $\mathrm{HD}(p,q)$ property. Theorem A implies – ignoring logarithmic factors and for general values of p and q – the existence of a hitting set of size no better than $O\left(p^d\right)$. Furthermore, it requires $q \geq d+1$ – a necessary condition for arbitrary convex objects[2] but not for balls.

Almost 60 years ago, Danzer [4, 5] considered the $\mathrm{HD}(p,q)$ problem for balls. The best bound that we are aware of, derived from the survey of Eckhoff [7] by combining inequalities (4.2), (4.4) and (4.5), is stated below. It is better than the one from Theorem A quantitatively, but also in that it gives a bound requiring only that $q \geq 2$. Further, for a very specific case – namely when $p = q$ and $(d-q)$ is $O(\log d)$ – it succeeds in giving polynomial bounds.

▶ **Theorem B** ([7]). *Let $\mathcal{B}$ be a finite set of balls in $\mathbb{R}^d$. If $\mathcal{B}$ satisfies the* $\mathrm{HD}(p,q)$ *property for some $d \geq p \geq q \geq 2$, then there exists a hitting set for $\mathcal{B}$ of size at most*

$$\sqrt{\frac{3\pi}{2}} \cdot 2^{d-q} \cdot \left((p-q) \cdot 2^q \cdot d^{\frac{3}{2}} \cdot g(d) + 4\,(d-q+2)^{\frac{3}{2}} \cdot g(d-q+2)\right)$$

where $g(x) = \log x + \log\log x + 1$. *Ignoring logarithmic terms, the above bound is of the form* $\Theta\left((p-q) \cdot 2^d \cdot d^{\frac{3}{2}} + 2^{d-q} \cdot (d-q)^{\frac{3}{2}}\right)$. *If $p \neq q$ the first term dominates, otherwise the second term dominates.*

Turning towards the lower-bound for the case where $\mathcal{C}$ is a set of unit balls in $\mathbb{R}^d$, Bourgain and Lindenstrauss [2] proved a lower-bound of 1.0645^d when $p = q = 2$ in $\mathbb{R}^d$, i.e., one needs at least 1.0645^d points to hit all pairwise intersecting unit balls in $\mathbb{R}^d$.

Our Result

We consider a more general set up for the $\mathrm{HD}(p,q)$ problem, as follows.

Let C be a convex object in $\mathbb{R}^d$, and P a set of n points lying outside C. For each $p \in P$, let H_p be the set of hyperplanes separating p from C. Let C_p be the set of points in $\mathbb{R}^d$ *dual* to the hyperplanes in H_p (see [12, Chapter 5.1]), and let $\mathcal{S} = \{C_p \colon p \in P\}$.

Our goal is to study the $\mathrm{HD}(p,q)$ property for $\mathcal{S}$ – namely, that out of every p objects of $\mathcal{S}$, there exists a point in $\mathbb{R}^d$ common to at least q of them. This is equivalent to the property of C and P that out of every p-sized set $P' \subseteq P$, there exists a hyperplane separating C from a q-sized subset $P'' \subset P'$ – or equivalently, $\mathrm{conv}(P'')$ is disjoint from C.

Our main theorem is the following. For a simpler expression, let c_q, c_p be two positive integers such that $p = c_p + \left\lfloor \frac{d}{2} \right\rfloor$ and $q = c_q + \left\lfloor \frac{d}{2} \right\rfloor$.

[2] There are easy examples, e.g. when the convex objects are hyperplanes in $\mathbb{R}^d$.

▶ **Theorem 1.** *Let C be a bounded convex object in $\mathbb{R}^d$ and P a set of n points lying outside C. Further let c_p, c_q be two integers, with $1 \leq c_q \leq c_p \leq n - \lfloor \frac{d}{2} \rfloor$, such that for every $c_p + \lfloor \frac{d}{2} \rfloor$ points of P, there exists a subset of size $c_q + \lfloor \frac{d}{2} \rfloor$ whose convex-hull is disjoint from C. Then the points of P can be partitioned into*

$$\lambda_d(c_p, c_q) = K_2 \frac{d}{c_q} \cdot \left(\sqrt{2}K_1\right)^{\frac{d}{c_q}} \cdot (\lfloor d/2 \rfloor + c_q)^2 \cdot (\lfloor d/2 \rfloor + c_p)^{1+\frac{\lfloor d/2 \rfloor - 1}{c_q}} \cdot \log(\lfloor d/2 \rfloor + c_p)$$

sets, each of whose convex-hull is disjoint from C. Here K_1, K_2 are absolute constants independent of n, d, c_p and c_q. Furthermore, such a partition can be computed in polynomial time.

The proof, presented in Section 2, is a combination of three ingredients: the Alon-Kleitman technique [1], bounds on independent sets in hypergraphs [9] and bounds on $(\leq k)$-sets for half-spaces [3]. It is an extension of the proof in [14] which studied Carathéodory's theorem in this setting.

▶ Remark. The restriction that $q \geq \lfloor \frac{d}{2} \rfloor + 1$ is necessary – as can be seen when P form the vertices of a cyclic polytope in $\mathbb{R}^d$ and C is a slightly shrunk copy of $\text{conv}(P)$.

▶ Remark. Note that when $c_q \geq \beta \cdot \frac{d}{2}$ for any absolute constant $\beta > 0$, the above bound is *polynomial* in the dimension d – it is upper-bounded by $O\left(q^2 p^{1+\frac{1}{\beta}} \log p\right)$.

▶ Remark. It was shown in [13] that C_p is a convex object in $\mathbb{R}^d$ and thus the bounds of Theorem A apply. As before, Theorem 1 substantially improves upon this, as the bounds following from Theorem A are exponential in d and furthermore, require $q \geq d + 1$.

As an immediate corollary of Theorem 1, we obtain the first improvements to the old bound on the (p, q) problem for balls in $\mathbb{R}^d$. The bound in Theorem B is exponential in d – except in special cases where $p = q$ *and* $(d - q)$ is[3] $O(\log d)$. On the other hand, our result gives polynomial bounds as long as $q \geq \beta d$ for any constant $\beta > \frac{1}{2}$.

▶ **Corollary 2** (Hadwiger-Debrunner (p, q) bound for balls in $\mathbb{R}^d$). *Let $\mathcal{B}$ be collection of balls in $\mathbb{R}^d$ such that for every subset of $c_p + \lfloor \frac{d+1}{2} \rfloor$ balls in $\mathcal{B}$, some $c_q + \lfloor \frac{d+1}{2} \rfloor$ have a common intersection, where c_p and c_q are integers such that $1 \leq c_q \leq c_p \leq n - \lfloor \frac{d+1}{2} \rfloor$. Then there exists a set X of $\lambda_{d+1}(c_p, c_q)$ points that form a hitting set for the balls in $\mathcal{B}$. Here $\lambda_{d+1}(\cdot, \cdot)$ is the function defined in the statement of Theorem 1.*

Proof. Observe that one can stereographically 'lift' balls in $\mathbb{R}^d$ to caps of a sphere S in $\mathbb{R}^{d+1}$, where a cap of a sphere is a portion of the sphere contained in a half-space that doesn't contain the center of the sphere. Thus we will prove a slightly more general result where $\mathcal{B}$ consists of caps of a d-dimensional sphere S embedded in $\mathbb{R}^{d+1}$.

For a point $x \in S$, let h_x denote the hyperplane tangent to S at x. For any point y lying outside S, define the *separating set of* y to be

$$S_y = \{z \in S \colon h_z \text{ separates } y \text{ and } S\}.$$

Geometrically, S_y is the set of points of S 'visible' from y, and form a cap of S. Furthermore, for any cap K of S, there is a unique point w such that $K = S_w$. We denote this point w by $\text{apex}(K)$.

[3] Recall that Theorem B assumes $q \leq p \leq d$.

Given the set of caps $\mathcal{B}$ on S, consider the point set

$$\text{apex}\,(\mathcal{B}) = \{\text{apex}(B) \colon B \in \mathcal{B}\}\,.$$

Observe that for a point $x \in S$ and a cap $B \in \mathcal{B}$, $x \in B$ if and only if $x \in S_{\text{apex}(B)}$. As $\mathcal{B}$ satisfies the (p, q) property – namely that for every p-sized subset $\mathcal{B}'$ of $\mathcal{B}$, there exists a point $x \in S$ lying in some q elements of $\mathcal{B}'$ – we have that for every p-sized subset A' of apex$(\mathcal{B})$, there exists a point $x \in S$ lying in the separating set of some q points of A'. In other words, h_x separates these q points from S.

Applying Theorem 1 with $C = S$ and $P = \text{apex}\,(\mathcal{B})$ in dimension $d+1$, we conclude that P can be partitioned into a family Ξ of $\lambda_{d+1}(c_p, c_q)$ sets, each of whose convex hull is disjoint from S. Consider a set $P' \in \Xi$. Since the convex hull of P' is disjoint from S, we can find a hyperplane h_x tangent to S at x such that h_x separates P' from S. This implies that all the caps in $\mathcal{B}$ corresponding to the points in P' contain the point x. Thus for each set of Ξ we obtain a point which is contained in all the caps corresponding to the points in that set. These $|X| = \lambda_{d+1}(c_p, c_q)$ points form the required set X. ◀

Our results complement the recent results of Keller, Smorodinsky and Tardos [9, 10] who obtain polynomial bounds for regions of low union complexity in the plane.

2 Proof of Theorem 1

Given a set P of points in $\mathbb{R}^d$ and an integer $k \geq 1$, a set $P' \subseteq P$ is called a k-set of P if $|P'| = k$ and if there exists a half-space h in $\mathbb{R}^d$ such that $P' = P \cap h$. A set $P' \subseteq P$ is called a $(\leq k)$-set if P' is a l-set for some $l \leq k$. The next well-known theorem gives an upper-bound on the number of $(\leq k)$-sets in a point set (see [17]).

▶ **Theorem 3** (Clarkson-Shor [3]). *For any integer $k \geq \left\lfloor \frac{d}{2} \right\rfloor + 1$, the number of $(\leq k)$-sets of any set of n points in $\mathbb{R}^d$ is at most*

$$\kappa_d\,(n, k) = 2\left(\frac{K_1}{\lceil d/2 \rceil}\right)^{\lceil d/2 \rceil} \binom{n}{\lfloor d/2 \rfloor} (k + \lceil d/2 \rceil)^{\lceil d/2 \rceil} \;\leq\; \kappa'_d\,(k) \cdot n^{\lfloor d/2 \rfloor}, \tag{1}$$

where $\kappa'_d\,(k) = 2K_1^d \lfloor d/2 \rfloor^{-\lfloor d/2 \rfloor} \left(1 + \frac{k}{\lceil d/2 \rceil}\right)^{\lceil d/2 \rceil}$ and $K_1 \geq e$ is an absolute constant independent of n, d and k.

▶ **Definition 4** (Depth). Given a set P of n points in $\mathbb{R}^d$ and any set $Q \subseteq P$, define the *depth* of Q with respect to P, denoted depth$_P(Q)$, to be the minimum number of points of P contained in any half-space containing Q.

For two parameters $l \geq k \geq 2$, let $\tau_d\,(n, k, l)$ denote the maximum number of subsets of size k and depth at most l with respect to P in any set P of n points in $\mathbb{R}^d$:

$$\tau_d\,(n, k, l) = \max_{\substack{P \subseteq \mathbb{R}^d \\ |P| = n}} |\{Q \subseteq P \colon |Q| = k \text{ and } \text{depth}_P\,(Q) \leq l\}|\,.$$

The following statement is easily implied by an application of the Clarkson-Shor technique [3] (e.g., see [16]).

▶ **Theorem 5.** *For parameters $l \geq k \geq \left\lfloor \frac{d}{2} \right\rfloor + 1$,*

$$\tau_d(n, k, l) \leq e \cdot \kappa_d(n, k) \cdot l^{k - \lfloor d/2 \rfloor},$$

where the function $\kappa(\cdot, \cdot)$ is as defined in Equation (1).

Proof. Let P be any set of n points in $\mathbb{R}^d$. Let t be the number of sets of P of size k and depth at most l. Pick each element of P independently with probability $\rho = \frac{1}{l}$ to get a random sample R. The expected number of k-sets in R satisfies

$$\begin{aligned}
\rho^k \cdot (1-\rho)^{l-k} \cdot t &\leq \mathbb{E}[\text{ number of } k\text{-sets in } R\ ] \\
&\leq 2\left(\frac{K_1}{\lceil d/2\rceil}\right)^{\lceil \frac{d}{2}\rceil} \mathbb{E}\left[\binom{|R|}{\lfloor \frac{d}{2}\rfloor}\right]\left(k+\left\lceil\frac{d}{2}\right\rceil\right)^{\lceil \frac{d}{2}\rceil} \\
&= 2\left(\frac{K_1}{\lceil d/2\rceil}\right)^{\lceil \frac{d}{2}\rceil} \binom{n}{\lfloor \frac{d}{2}\rfloor}\rho^{\lfloor \frac{d}{2}\rfloor}\left(k+\left\lceil\frac{d}{2}\right\rceil\right)^{\lceil \frac{d}{2}\rceil} \\
&= \kappa_d(n,k)\cdot \rho^{\lfloor \frac{d}{2}\rfloor} \\
\implies t &\leq \frac{\kappa_d(n,k)\cdot\rho^{\lfloor \frac{d}{2}\rfloor}}{\rho^k\cdot(1-\rho)^{l-k}} \leq e\cdot\kappa_d(n,k)\cdot l^{k-\lfloor d/2\rfloor},
\end{aligned}$$

as $\left(1-\frac{1}{l}\right)^{-(l-k)} \leq e$ for any $l \geq k \geq 2$. ◀

▶ **Lemma 6.** *Let C be a bounded convex object in $\mathbb{R}^d$, and P a set of n points lying outside C. Let $p \geq q \geq \lfloor \frac{d}{2}\rfloor + 1$ be parameters such that for every subset $Q \subseteq P$ of size p, there exists a set $Q' \subset Q$ of size q such that Q' can be separated from C by a hyperplane. Then there exists a hyperplane separating at least*

$$\left(2\,q\,p^{q-1}\cdot e\,\kappa'_d(q)\right)^{\frac{1}{\lfloor d/2\rfloor - q}}$$

fraction of the points of P from C.

Proof. From [6, 9], it follows that the number of distinct q-tuples of P that can be separated from C by a hyperplane is, assuming that $n \geq 2p$, at least

$$\frac{n-p+1}{n-q+1}\frac{\binom{n}{q}}{\binom{p-1}{q-1}} \geq \frac{n^q}{2q\,p^{q-1}}.$$

Let l be the maximum depth (Definition 4) of any of these separable q-tuples. The number of such tuples is therefore at most $\tau_d(n,q,l)$. Thus by Theorem 5 we must have

$$\frac{n^q}{2q\,p^{q-1}} \leq \tau_d\,(n,q,l) \leq e\,\kappa_d\,(n,q)\ l^{q-\lfloor d/2\rfloor}.$$

Re-arranging the terms and from inequality (1), we get

$$\begin{aligned}
l \geq \left(\frac{n^q}{2\,q\,p^{q-1}\cdot e\,\kappa_d\,(n,q)}\right)^{\frac{1}{q-\lfloor d/2\rfloor}} &\geq \left(\frac{n^q}{2\,q\,p^{q-1}\cdot e\,\kappa'_d\,(q)\ n^{\lfloor \frac{d}{2}\rfloor}}\right)^{\frac{1}{q-\lfloor d/2\rfloor}} \\
&= n\cdot\left(2\,q\,p^{q-1}\cdot e\,\kappa'_d\,(q)\right)^{\frac{1}{\lfloor d/2\rfloor - q}}.
\end{aligned}$$

Thus one of the separable q-tuples, say $P' \subseteq P$, must have depth at least l; in other words, the hyperplane separating P' from C must contain at least l points of P. This is the required hyperplane. ◀

We now prove a weighted version of the above statement.

▶ **Corollary 7.** *Let C be a bounded convex object in $\mathbb{R}^d$, and P a weighted set of n points lying outside C, where the weight of each point $p \in P$ is a non-negative rational number. Let $p \geq q \geq \left\lfloor \frac{d}{2} \right\rfloor + 1$ be parameters such that for every subset $Q \subseteq P$ of size p, there exists a set $Q' \subset Q$ of size q such that Q' can be separated from C by a hyperplane. Then there exists a hyperplane separating a set of points whose weight is at least*

$$\alpha_d(p,q) = \left(2e\,\kappa_d'(q)\; q^q\, p^{q-1}\right)^{\frac{1}{\lfloor d/2 \rfloor - q}}$$

fraction of the total weight of the points in P.

Proof. By appropriately scaling all the rational weights, we may assume that each weight is a non-negative integer and we replace a point with weight m by m unweighted copies of the point. Let P' be the new set of points. Observe that any set S of pq points in P' either contains q copies of some point in P or it contains p distinct points from P. In either case, there is hyperplane separating q points of S from C. Thus, we can apply Lemma 6 to the point set P' with the parameter p in the lemma replaced by pq. The result follows. ◀

Finally we return to the proof of the main theorem.

▶ **Theorem 1.** *Let C be a bounded convex object in $\mathbb{R}^d$ and P a set of n points lying outside C. Further let c_p, c_q be two integers, with $1 \leq c_q \leq c_p \leq n - \left\lfloor \frac{d}{2} \right\rfloor$, such that for every $c_p + \left\lfloor \frac{d}{2} \right\rfloor$ points of P, there exists a subset of size $c_q + \left\lfloor \frac{d}{2} \right\rfloor$ whose convex-hull is disjoint from C. Then the points of P can be partitioned into*

$$\lambda_d\left(c_p, c_q\right) = K_2\,\frac{d}{c_q}\cdot\left(\sqrt{2}K_1\right)^{\frac{d}{c_q}}\cdot\left(\lfloor d/2 \rfloor + c_q\right)^2\;\cdot\left(\lfloor d/2 \rfloor + c_p\right)^{1+\frac{\lfloor d/2 \rfloor - 1}{c_q}}\cdot\log\left(\lfloor d/2 \rfloor + c_p\right)$$

sets, each of whose convex-hull is disjoint from C. Here K_1, K_2 are absolute constants independent of n, d, c_p and c_q. Furthermore, such a partition can be computed in polynomial time.

Proof. Let $p = c_p + \lfloor d/2 \rfloor$ and $q = c_q + \lfloor d/2 \rfloor$. Let $\mathcal{H}$ be the set of all hyperplanes separating a distinct subset of points of P from C. As the number of subsets of P is finite, one can assume that $\mathcal{H}$ is also finite. Consider the following linear program on $|\mathcal{H}|$ variables $\{u_h \geq 0\colon h \in \mathcal{H}\}$:

$$\min \sum_{h\in\mathcal{H}} u_h, \quad \text{such that} \quad \forall r \in P\colon \sum_{\substack{h\in\mathcal{H} \\ h \text{ separates } r \text{ from } C}} u_h \geq 1. \tag{2}$$

The LP-dual to the above program, on $|P|$ variables $\{w_r \geq 0\colon r \in P\}$, is:

$$\max \sum_{p\in P} w_p, \quad \text{such that} \quad \forall h \in \mathcal{H}\colon \sum_{\substack{r\in P \\ h \text{ separates } r \text{ from } C}} w_r \leq 1. \tag{3}$$

Consider an optimal solution w^* of the dual linear program and interpret w_r^* as the weight of each $r \in P$. Since the weights are rational, by Corollary 7, there exists a hyperplane $h \in \mathcal{H}$ separating a subset of P of combined weight at least $\epsilon = \alpha_d(p,q)$ fraction of the total weight of all the points. Since the total weight of the points in any half-space is constrained to be at most 1 by the linear program, the total weight of all the points of P must be at most $\frac{1}{\epsilon}$. In other words, the optimal value of linear program (3) is at most $\frac{1}{\epsilon}$. Since the optimal values of both linear programs are equal due to strong duality, the optimal value of linear program (2) is also at most $\frac{1}{\epsilon}$.

Let u^* be the optimal solution of linear program (2). If we interpret u_h as the weight of the hyperplane h, the constraints of the program imply that each point is separated by a set of hyperplanes in $\mathcal{H}$ whose combined weight is at least 1 out of a total weight of at most $\frac{1}{\epsilon}$ – in other words, at least ϵ-th fraction of the total weight of $\mathcal{H}$. By associating with each hyperplane the half-space bounded by it and not containing C, and using the ϵ-net theorem for half-spaces in $\mathbb{R}^d$ (see [11]), there exists a set of $O\left(\frac{d}{\epsilon}\log\frac{1}{\epsilon}\right)$ hyperplanes which together separate all points of P from C. Recalling that

$$\frac{1}{\epsilon} = \frac{1}{\alpha_d(p,q)} = \left(2e\,\kappa_d'(q)\,q^q\,p^{q-1}\right)^{\frac{1}{q-\lfloor d/2\rfloor}} = \left(2e\,\kappa_d'(q)\,q^q\,p^{q-1}\right)^{\frac{1}{c_q}}.$$

and that $\kappa_d'(q) = 2K_1^d \lfloor d/2\rfloor^{-\lfloor d/2\rfloor}\left(1+\frac{q}{\lceil d/2\rceil}\right)^{\lceil d/2\rceil}$, we get

$$\begin{aligned}
\frac{1}{\epsilon} &= \left(4K_1^d e \lfloor d/2\rfloor^{-\lfloor d/2\rfloor}\left(1+\frac{q}{\lceil d/2\rceil}\right)^{\lceil d/2\rceil} q^q\,p^{q-1}\right)^{\frac{1}{c_q}}\\
&\le \left(4K_1^{d+1}\lfloor d/2\rfloor^{-d}\,(c_q+d)^{\lceil d/2\rceil}\,q^q\,p^{q-1}\right)^{\frac{1}{c_q}} \qquad (\text{using } e \le K_1 \text{ and } q = c_q + \lfloor d/2\rfloor)\\
&\le \left(4K_1^{d+1}\lfloor d/2\rfloor^{-d}(c_q+d)^{\lceil d/2\rceil} q^{c_q+\lfloor d/2\rfloor} p^{c_q+\lfloor d/2\rfloor-1}\right)^{\frac{1}{c_q}}\\
&= O\left(K_1^{\frac{d}{c_q}}\,\lfloor d/2\rfloor^{-\frac{d}{c_q}}\,(c_q+d)^{\frac{\lceil d/2\rceil}{c_q}}\,(c_q+\lfloor d/2\rfloor)^{1+\frac{\lfloor d/2\rfloor}{c_q}}\,(c_p+\lfloor d/2\rfloor)^{1+\frac{\lfloor d/2\rfloor-1}{c_q}}\right)\\
&= O\left(K_1^{\frac{d}{c_q}}\,d^{2+\frac{\lfloor d/2\rfloor-1}{c_q}}\,\left(1+\frac{c_q}{d}\right)^{\frac{\lceil d/2\rceil}{c_q}}\,\left(1+\frac{c_q}{\lfloor d/2\rfloor}\right)^{1+\frac{\lfloor d/2\rfloor}{c_q}}\,\left(1+\frac{c_p}{\lfloor d/2\rfloor}\right)^{1+\frac{\lfloor d/2\rfloor-1}{c_q}}\right)\\
&= O\left(K_1^{\frac{d}{c_q}}\,d^{2+\frac{\lfloor d/2\rfloor-1}{c_q}}\,e^{\frac{c_q}{d}\cdot\frac{\lceil d/2\rceil}{c_q}}\,\left(1+\frac{c_q}{\lfloor d/2\rfloor}\right)\,e^{\frac{c_q}{\lfloor d/2\rfloor}\cdot\frac{\lfloor d/2\rfloor}{c_q}}\,\left(1+\frac{c_p}{\lfloor d/2\rfloor}\right)^{1+\frac{\lfloor d/2\rfloor-1}{c_q}}\right)\\
&= O\left(K_1^{\frac{d}{c_q}}\,d^{2+\frac{\lfloor d/2\rfloor-1}{c_q}}\,\left(1+\frac{c_q}{\lfloor d/2\rfloor}\right)\,\left(1+\frac{c_p}{\lfloor d/2\rfloor}\right)^{1+\frac{\lfloor d/2\rfloor-1}{c_q}}\right)\\
&= O\left(K_1^{\frac{d}{c_q}}\,2^{\frac{d}{2c_q}}\,\left(\lfloor d/2\rfloor+c_q\right)\,\left(\lfloor d/2\rfloor+c_p\right)^{1+\frac{\lfloor d/2\rfloor-1}{c_q}}\right)\\
&= O\left(\left(\sqrt{2}K_1\right)^{\frac{d}{c_q}}\,\left(\lfloor d/2\rfloor+c_q\right)\,\left(\lfloor d/2\rfloor+c_p\right)^{1+\frac{\lfloor d/2\rfloor-1}{c_q}}\right).
\end{aligned}$$

The Big-Oh notation here does not hide dependencies on d – namely we do not treat d as a constant. From the above it follows that

$$\log\frac{1}{\epsilon} = O\left(c_q^{-1}\left(\lfloor d/2\rfloor+c_q\right)\log\left(\lfloor d/2\rfloor+c_p\right)\right).$$

Thus, $\frac{d}{\epsilon}\log\frac{1}{\epsilon}$ is

$$O\left(d\cdot\left(\left(\sqrt{2}K_1\right)^{\frac{d}{c_q}}\,\left(\lfloor d/2\rfloor+c_q\right)\,\left(\lfloor d/2\rfloor+c_p\right)^{1+\frac{\lfloor d/2\rfloor-1}{c_q}}\right)\cdot\left(c_q^{-1}\left(\lfloor d/2\rfloor+c_q\right)\log\left(\lfloor d/2\rfloor+c_p\right)\right)\right)$$

which simplifies to

$$O\left(\frac{d}{c_q}\left(\sqrt{2}K_1\right)^{\frac{d}{c_q}}\left(\lfloor d/2\rfloor+c_q\right)^2\,\left(\lfloor d/2\rfloor+c_p\right)^{1+\frac{\lfloor d/2\rfloor-1}{c_q}}\,\log(\lfloor d/2\rfloor+c_p)\right).$$

Since linear programs can be solved in polynomial time and epsilon nets can be computed in polynomial time, the partition of P into the above number of sets can be achieved in polynomial time. The theorem follows. ◀

References

1 N. Alon and D. Kleitman. Piercing convex sets and the Hadwiger–Debrunner (p, q)-problem. *Adv. Math.*, 96(1):103–112, 1992.

2 J. Bourgain and J. Lindenstrauss. On covering a set in R^n by balls of the same diameter. In *Geometric Aspects of Functional Analysis*, pages 138–144. Springer Berlin Heidelberg, 1991.

3 K. Clarkson and P. Shor. Applications of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4(5):387–421, 1989.

4 L. Danzer. *Uber zwei Lagerungsprobleme; Abwandlungen einer Vermutung von T. Gallai*. PhD thesis, Techn. Hochschule Munchen, 1960.

5 L. Danzer. Uber durchschnittseigenschaften n-dimensionaler kugelfamilien. *J. Reine Angew. Math.*, 208:181–203, 1961.

6 D. de Caen. Extension of a theorem of Moon and Moser on complete subgraphs. *Ars Combin.*, 16:5–10, 1983.

7 J. Eckhoff. A survey of the Hadwiger-Debrunner (p, q)-problem. In *Discrete and Computational Geometry: The Goodman-Pollack Festschrift*, pages 347–377. Springer, 2003.

8 A. Holmsen and R. Wenger. Helly-type theorems and geometric transversals. In J. E. Goodman, J. O'Rourke, and C. D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, pages 91–123. CRC Press LLC, 2017.

9 C. Keller, S. Smorodinsky, and G. Tardos. Improved bounds on the Hadwiger-Debrunner numbers. *Israel J. of Math., to appear*, 2017.

10 C. Keller, S. Smorodinsky, and G. Tardos. On max-clique for intersection graphs of sets and the hadwiger-debrunner numbers. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2254–2263, 2017.

11 A. Kupavskii, N. H. Mustafa, and J. Pach. Near-optimal lower bounds for ϵ-nets for half-spaces and low complexity set systems. In Martin Loebl, Jaroslav Nešetřil, and Robin Thomas, editors, *A Journey Through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 527–541. Springer International Publishing, 2017.

12 J. Matoušek. *Lectures in Discrete Geometry.* Springer-Verlag, New York, NY, 2002.

13 N. H. Mustafa and S. Ray. Weak ϵ-nets have a basis of size $O(1/\epsilon \log 1/\epsilon)$. *Comp. Geom: Theory and Appl.*, 40(1):84–91, 2008.

14 N. H. Mustafa and S. Ray. An optimal generalization of the colorful Carathéodory theorem. *Discrete Mathematics*, 339(4):1300–1305, 2016.

15 N. H. Mustafa and K. Varadarajan. Epsilon-approximations and epsilon-nets. In J. E. Goodman, J. O'Rourke, and C. D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, pages 1241–1268. CRC Press LLC, 2017.

16 S. Smorodinsky, M. Sulovský, and U. Wagner. On center regions and balls containing many points. In *Proceedings of the 14th annual International Conference on Computing and Combinatorics*, COCOON '08, pages 363–373, 2008.

17 U. Wagner. k-sets and k-facets. In J.E. Goodman, J. Pach, and R. Pollack, editors, *Surveys on Discrete and Computational Geometry: Twenty Years Later*, Contemporary Mathematics, pages 231–255. American Mathematical Society, 2008.

Quasi-Polynomial Time Approximation Schemes for Packing and Covering Problems in Planar Graphs

Michał Pilipczuk[1]
Institute of Informatics, University of Warsaw, Poland
michal.pilipczuk@mimuw.edu.pl

Erik Jan van Leeuwen
Department of Information and Computing Sciences, Utrecht University, The Netherlands
e.j.vanleeuwen@uu.nl

Andreas Wiese[2]
Department of Industrial Engineering and Center for Mathematical Modeling, Universidad de Chile, Chile
awiese@dii.uchile.cl

Abstract

We consider two optimization problems in planar graphs. In Maximum Weight Independent Set of Objects we are given a graph G and a family $\mathcal{D}$ of *objects*, each being a connected subgraph of G with a prescribed weight, and the task is to find a maximum-weight subfamily of $\mathcal{D}$ consisting of pairwise disjoint objects. In Minimum Weight Distance Set Cover we are given an edge-weighted graph G, two sets $\mathcal{D}, \mathcal{C}$ of vertices of G, where vertices of $\mathcal{D}$ have prescribed weights, and a nonnegative radius r. The task is to find a minimum-weight subset of $\mathcal{D}$ such that every vertex of $\mathcal{C}$ is at distance at most r from some selected vertex. Via simple reductions, these two problems generalize a number of geometric optimization tasks, notably Maximum Weight Independent Set for polygons in the plane and Weighted Geometric Set Cover for unit disks and unit squares. We present *quasi-polynomial time approximation schemes* (QPTASs) for both of the above problems in planar graphs: given an accuracy parameter $\epsilon > 0$ we can compute a solution whose weight is within multiplicative factor of $(1 + \epsilon)$ from the optimum in time $2^{\mathrm{poly}(1/\epsilon, \log |\mathcal{D}|)} \cdot n^{\mathcal{O}(1)}$, where n is the number of vertices of the input graph. Our main technical contribution is to transfer the techniques used for recursive approximation schemes for geometric problems due to Adamaszek, Har-Peled, and Wiese [1, 2, 4] to the setting of planar graphs. In particular, this yields a purely combinatorial viewpoint on these methods.

2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases QPTAS, planar graphs, Voronoi diagram

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.65

Related Version A full version of the paper is available at `http://arxiv.org/abs/1807.07626`.

Acknowledgements We thank Dániel Marx for insightful discussions on the approach to optimization problems in geometric and planar settings via Voronoi diagrams.

[1] The research of Mi. Pilipczuk was partially supported by Polish National Science Centre grant UMO-2013/11/D/ST6/03073, and is a part of project TOTAL that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 677651).
[2] The research of A. Wiese is supported by the Millennium Nucleus Information and Coordination in Networks ICM/FIC RC130003 and by the grant Fondecyt Regular 1170223.

© Michał Pilipczuk, Erik Jan van Leeuwen, and Andreas Wiese;
licensed under Creative Commons License CC-BY

1 Introduction

Independent Set and Dominating Set are fundamental optimization problems on graphs. Given a graph G where each vertex v has a weight $\mathbf{w}(v)$, in Independent Set one seeks to find a vertex subset $I \subseteq V(G)$ of maximum possible weight such that no two vertices in I are adjacent, whereas in Dominating Set one searches for a vertex subset D of minimum possible weight such that each vertex $v \in V(G)$ is contained in D or adjacent to a vertex in D. Even in the unit-weight setting, both problems are notoriously hard to approximate and they are also W[1]-hard, i.e., we do not expect that they admit *fixed-parameter tractable* (*fpt*) algorithms running in time $f(k) \cdot n^{\mathcal{O}(1)}$, where k is the expected solution size.

Therefore, special cases of the problems were investigated, for instance the case where the input graph is planar. On planar graphs, classic layering techniques can be applied to show that both problems admit EPTASs, i.e., $(1+\epsilon)$-approximation algorithms with a running time of $f(1/\epsilon)n^{\mathcal{O}(1)}$ for some function f, and fpt algorithms for the parameterization by the solution size, i.e., for a parameter k, algorithms running in time $f(k)n^{\mathcal{O}(1)}$ that find a best solution among those of size at most k. Given these results, it is natural to consider generalizations of the above problems on planar graphs.

In this paper we study the Distance Independent Set and the Distance Dominating Set problems. Given additionally a value $r \in \mathbb{R}$ and weights on the edges of G, in the Distance Independent Set problem we require that any two selected vertices in I are at distance larger than r from each other, and in the Distance Dominating Set problem we require that each vertex $v \in V(G)$ is at distance at most r from some vertex of D. Let us stress that we assume that r is part of the input and in particular not assumed to be a constant; in fact, for constant r and unit edge weights, it is well-known that the same layering techniques easily yield EPTASs and fpt algorithms on planar graphs. In the parameterized setting, both problems are W[1]-hard even for unit weights; however, the trivial $n^{\mathcal{O}(k)}$-time algorithms can be improved to $n^{\mathcal{O}(\sqrt{k})}$-time algorithms [5]. These parameterized algorithms extend a technique originally developed to design quasi-polynomial time approximation schemes (QPTASs) for Independent Set and Dominating Set in the geometric (Euclidean) setting [1, 2, 4]. The idea is to guess a sparse separator that has only small intersection with the optimal solution and that splits the problem into two roughly equal-sized subproblems, and then to solve the subproblems recursively. The natural question arises whether one can transfer the insights obtained in the parameterized setting back to approximation algorithms, and obtain approximation schemes for Distance Independent Set and Distance Dominating Set in planar graphs.

Our contribution. In this paper we show that this is indeed possible and we present the first quasi-polynomial time approximation schemes for Distance Independent Set and Distance Dominating Set on planar graphs when r is part of the input. In fact, we give QPTASs for two even more general problems, which we name Maximum Weight Independent Set of Objects (MWISO) and Minimum Weight Distance Set Cover (MWDSC). In MWISO we are given a graph G and a family $\mathcal{D}$ of *objects*, each being a connected subgraph of G with a prescribed weight, and the goal is to find a maximum-weight subfamily of $\mathcal{D}$ consisting of pairwise disjoint objects. In MWDSC we are given an edge-weighted graph G, subsets of vertices $\mathcal{D}$ and $\mathcal{C}$ where vertices of $\mathcal{D}$ are weighted, and radius $r \in \mathbb{R}$. The goal is to find a minimum-weight subset $\mathcal{F} \subseteq \mathcal{D}$ that r-*covers* $\mathcal{C}$ in the sense that each vertex of $\mathcal{C}$ is at distance at most r from some vertex of $\mathcal{F}$. MWISO generalizes Distance Independent Set by taking $\mathcal{D}$ to be the family $\{\{v\colon \mathrm{dist}(u,v) \le r/2\}\colon u \in V(G)\}$ of all balls of radius $r/2$ in the graph, while MWDSC generalizes Distance Dominating Set by taking $\mathcal{C} = V(G)$. The following statements summarize our results.

▶ **Theorem 1.** *The* Maximum Weight Independent Set of Objects *problem in planar graphs admits a QPTAS with running time $2^{\mathrm{poly}(1/\epsilon, \log N)} \cdot n^{\mathcal{O}(1)}$, where n is the vertex count of the input graph and $N = |\mathcal{D}|$ is the number of objects in the input.*

▶ **Theorem 2.** *The* Minimum Weight Distance Set Cover *problem in planar graphs admits a QPTAS with running time $2^{\mathrm{poly}(1/\epsilon, \log N)} \cdot n^{\mathcal{O}(1)}$, where n is the vertex count of the input graph and $N = |\mathcal{D}|$ is the number of vertices allowed to be selected to the solution.*

To obtain our QPTASs for MWISO, we extend the machinery developed in [1, 2, 4] for optimization problems in geometric settings to problems in planar graphs. The heart of our technical contribution is to show that for any instance of the above problems there is a set of candidate separators of polynomial size such that one of them splits the given problem in a balanced way and intersects only a tiny fraction of the given solution. The latter is important since the intersected objects will be lost (in the case of MWISO) or might be paid twice (in the case of MWDSC) and hence we need to bound their total weight by ϵOPT. We state here an informal version of our separator lemma for the case of MWISO.

▶ **Lemma 3** (Informal)**.** *In polynomial time we can compute a set $\mathbb{X} \subseteq 2^{\mathcal{D}}$ of separators such that for every solution $\mathcal{F} \subseteq \mathcal{D}$, say of weight W, there exists $\mathcal{X} \in \mathbb{X}$ such that $\mathbf{w}(\mathcal{F} \cap \mathcal{X}) \leq \epsilon W$ and in the intersection graph of $\mathcal{D} - \mathcal{X}$ each connected component $\mathcal{C}$ satisfies $\mathbf{w}(\mathcal{C} \cap \mathcal{F}) \leq \frac{9}{10} W$.*

Using Lemma 3 as abstraction for finding separators, we can apply the same recursive scheme as [1, 2, 4]: we guess the correct separator $\mathcal{X} \in \mathbb{X}$, construct a subproblem for each connected component of the intersection graph of $\mathcal{D} - \mathcal{X}$, and recurse in each of them up to recursion depth $\mathcal{O}(\log |\mathcal{D}|)$. Thus, the only part of the reasoning that uses planarity is Lemma 3.

The proof of Lemma 3 follows the reasoning of Har-Peled [4]. The idea is to prove the following auxiliary result: for the optimal solution $\mathcal{F}$ (and in fact for any feasible solution) there exists a separator of length roughly $s = \mathcal{O}(\frac{1}{\epsilon} \ln \frac{1}{\epsilon})$ that cuts through at most an ϵ-fraction of the weight of $\mathcal{F}$ and splits the weight of $\mathcal{F}$ in a balanced way. Lemma 3 then follows by enumerating all candidates for such separators. In [4], the separator was simply a polygon with roughly s vertices. We lift this concept to planar graphs using *Voronoi separators* as in the work of Marx and Pilipczuk [5]. Intuitively, a Voronoi separator of length r is an alternating cyclic sequence of r objects from $\mathcal{D}$ and r faces of the graph, connected by shortest paths in order to form a closed curve; this curve splits the instance into two subinstances. Thus, shortest paths in the graph are the analogues of segments in the plane.

The auxiliary result is proved in [4] by showing that if $\mathcal{S}$ is a sample of size roughly s^2 from $\mathcal{F}$, where each object is sampled independently with probability proportional to its weight, then a balanced separator of length s in the Voronoi diagram of $\mathcal{S}$ satisfies all the required properties with high probability. We follow the same reasoning, however again we need to properly understand how geometric concept used in [4] – *spokes* and *corridors* – should be interpreted in planar graphs. Here, the technical toolbox for Voronoi diagrams and Voronoi separators developed in [5] becomes very useful. In particular, it turns out that a fine understanding of what faces are candidates for branching points of a Voronoi diagram, provided in [5], is essential to make the probabilistic argument work. Let us remark that we also somewhat simplify the original argument of Har-Peled by replacing the Exponential Decay Lemma with a direct probabilistic calculation.

To give the QPTAS for MWDSC we provide a variant of Lemma 3 suitable for this problem and then follow a similar recursive scheme as for Theorem 1. It is nice that we can reuse the above-mentioned auxiliary result introduced for Lemma 3 as a black-box, so the proof of the variant is relatively short. As in [7], the difference is that in the recursion instead of removing the guessed separator we preserve it in all the recursive subcalls, thus allowing double-buying objects from it. Due to space constraints, the proof of Theorem 2 is entirely deferred to the full version, while in this extended abstract we focus on proving Theorem 1.

Geometric problems. The above recursive machinery based on balanced separators was first introduced for obtaining a QPTAS for Maximum Weight Independent Set of Rectangles in the two-dimensional plane [1] and then extended for getting QPTASs for Maximum Weight Independent Set of Polygons [2, 4] and Weighted Geometric Set Cover (WGSR) for pseudo-disks [7]. We prove that Theorems 1 and 2 generalize these results, with the exception that for WGSR we can treat only the cases of unit disks and axis-parallel unit squares, instead of general families of pseudo-disks. In the full version we explain how to derive the mentioned results from our theorems.

2 Proof of the Separator Lemma for MWISO

In this section we prove the Separator Lemma for MWISO, which was informally stated as Lemma 3 and is formally stated below. For a family $\mathcal{D}$ of objects, $\mathrm{IntGraph}(\mathcal{D})$ denotes the *intersection graph* of $\mathcal{D}$: graph with vertex set $\mathcal{D}$ where two objects are adjacent iff they intersect. The reader may think of $\mathcal{F}$ being the optimal solution and of W being its weight.

▶ **Lemma 4** (Separator Lemma for MWISO). *Let G be an n-vertex planar graph and $\mathcal{D}$ be a weighted family of N objects in G. Let $0 < \epsilon < \frac{1}{10}$ and denote $s = 10^3 \cdot \frac{1}{\epsilon} \ln \frac{1}{\epsilon}$. Then there exists a family $\mathbb{X}$ consisting of subsets of $\mathcal{D}$ with the following properties:*

(A1) *$|\mathbb{X}| \le 6^{3s} N^{15s}$ and $\mathbb{X}$ can be computed in time $N^{\mathcal{O}(s)} \cdot n^{\mathcal{O}(1)}$; and*

(A2) *for every real $W \ge 0$ and subfamily $\mathcal{F} \subseteq \mathcal{D}$ of pairwise disjoint objects such that $\mathbf{w}(\mathcal{F}) \le W$ and $\mathbf{w}(p) \le s^{-2}W$ for each $p \in \mathcal{F}$, there exist $\mathcal{X} \in \mathbb{X}$ such that $\mathbf{w}(\mathcal{F} \cap \mathcal{X}) \le \epsilon W$ and for every connected component $\mathcal{C}$ of $\mathrm{IntGraph}(\mathcal{D}) \setminus \mathcal{X}$ we have $\mathbf{w}(\mathcal{C} \cap \mathcal{F}) \le \frac{9}{10}W$.*

The plan is as follows. We first recall the toolbox of *Voronoi separators*, introduced by Marx and Pilipczuk [5]. This allows us to state a stronger lemma, phrased as the existence of a short Voronoi separator appropriately breaking $\mathcal{F}$. We then show how Lemma 4 follows from this stronger result and subsequently prove the stronger result.

Before we proceed, let us set up the notation and basic assumptions about the input. Let G be the input graph. We assume the edges of G are assigned positive weights[3] so that we have the shortest-path metric in G: $\mathrm{dist}(u, v)$ denotes the shortest length of a path connecting u and v in G. We may assume that G is given with an embedding in a sphere Σ and that it is triangulated; that is, every face of G is a triangle. Indeed, adding edges of infinite weight to triangulate the graph neither distorts the metric nor spoils the connectivity of the objects. Also, for every face f of G we fix any its internal point c to be its *center*, and for each vertex u of f we fix some curve within f with endpoints u and c to be the *segment* connecting u and c so that segments connecting vertices of f with c pairwise do not cross.

We assume that shortest paths are unique and the distances between pairs of vertices are pairwise different: for every pair of vertices u, v there is a unique shortest path connecting u and v and for $\{u, v\} \neq \{u', v'\}$ we have $\mathrm{dist}(u, v) \neq \mathrm{dist}(u', v')$. This can be ensured by using lexicographic tie-breaking rules and it increases the running time only by polynomial factors.

[3] Obviously, edge weights are immaterial for the MWISO problem. However, it is convenient to think of G as edge-weighted so that we can define Voronoi diagrams. Furthermore, many results of this section will be reused for the MWDSC problem, where edge weights play a role in the problem.

We are given a family $\mathcal{D}$ of *objects* in G, where each object $p \in \mathcal{D}$ is a nonempty, connected subgraph of G. For any vertex u of G and any object $p \in \mathcal{D}$, let $\operatorname{dist}(u, p)$ be the length of the shortest path connecting u with any vertex of p. For each object $p \in \mathcal{D}$, we fix any spanning tree $T(p)$ of p. We also assume that the family $\mathcal{D}$ is weighted: every object $p \in \mathcal{D}$ is assigned a nonnegative real weight $\mathbf{w}(p)$. For $\mathcal{F} \subseteq \mathcal{D}$ we denote $\mathbf{w}(\mathcal{F}) = \sum_{p \in \mathcal{F}} \mathbf{w}(p)$.

2.1 Basic toolbox

Voronoi partitions and diagrams. A subfamily $\mathcal{F} \subseteq \mathcal{D}$ is *independent* if objects in $\mathcal{F}$ are pairwise vertex-disjoint. Such an independent subfamily $\mathcal{F}$ induces the *Voronoi partition* $\mathbb{M}_{\mathcal{F}}$, which is a partition of the vertex set of G into $|\mathcal{F}|$ parts according to the closest object from $\mathcal{F}$. Precisely, for $p \in \mathcal{F}$, we say that a vertex u belongs to the *Voronoi cell* $\mathbb{M}_{\mathcal{F}}(p)$ if $\operatorname{dist}(u, p) < \operatorname{dist}(u, p')$ for any $p' \in \mathcal{F}$, $p' \neq p$. Observe that ties do not happen due to distinctness of distances in G. We note that Marx and Pilipczuk consider in [5] a more general notion of a *normal subfamily*, but we do not need this generality here.

Assuming $|\mathcal{F}| \geq 4$, we define the *Voronoi diagram* induced by $\mathbb{M}_{\mathcal{F}}$ as follows. First, observe that every Voronoi cell $\mathbb{M}_{\mathcal{F}}(p)$, for $p \in \mathcal{F}$, induces a connected subgraph of G that contains p entirely (see Lemmas 4.1 and 4.2 in [5]). Extend $T(p)$ to a spanning tree $\widehat{T}(p)$ of $G[\mathbb{M}_{\mathcal{F}}(p)]$ by adding, for each vertex u of $\mathbb{M}_{\mathcal{F}}(p)$ that is not in p, the shortest path from u to p. Take the dual of G and remove all the edges dual to the edges of $\widehat{T}(p)$, for all $p \in \mathcal{F}$. Then exhaustively remove vertices of degree one, and finally replace each maximal path with internal vertices of degree 2 (so-called 2-*path*) by a single edge; the embedding of this edge is defined as the union of embeddings of edges of G comprising the original 2-path. Thus, we obtain a connected, 3-regular plane multigraph, called the *Voronoi diagram* of $\mathcal{F}$, whose faces bijectively correspond to the cells $\mathbb{M}_{\mathcal{F}}(p)$ for $p \in \mathcal{F}$. More precisely, every face of the Voronoi diagram of $\mathcal{F}$ is associated with a different object $p \in \mathcal{F}$ so that all the vertices of $\mathbb{M}_{\mathcal{F}}(p)$ are contained in this face. The 3-regularity of the diagram follows from the assumption that G is triangulated. From Euler's formula it follows that if $|\mathcal{F}| = k$, then H has k faces, $2k - 4$ vertices, and $3k - 6$ edges. See Lemmas 4.4 and 4.5 of [5] for a formal verification of these assertions, and Section 4.4 of [5] for a detailed description of the construction.

Branching points. If H is the Voronoi diagram of an independent subfamily $\mathcal{F} \subseteq \mathcal{D}$, then H is constructed from a subgraph of the dual of G by contracting maximal 2-paths. Hence, vertices of H correspond to faces of G. These primal faces, equivalently dual vertices, are called the *branching points* of the diagram H; intuitively, these are faces where the boundaries of Voronoi cells meet nontrivially. A priori, every face of G could be a branching point of the Voronoi diagram of some independent subfamily $\mathcal{F} \subseteq \mathcal{D}$. However, in [5] it is proved that the number of candidates for branching points can be bounded polynomially in $|\mathcal{D}|$.

▶ **Theorem 5** (Theorem 4.7 of [5]). *There exists a family I of faces of G with $|I| \leq |\mathcal{D}|^4$ such that the following holds: for every independent subfamily of objects $\mathcal{F} \subseteq \mathcal{D}$, all branching points of the Voronoi diagram of $\mathcal{F}$ are contained in I. Moreover, I can be computed in time polynomial in $|\mathcal{D}|$ and n.*

We fix the family I provided by Theorem 5 and call its members $\mathcal{D}$-*important* faces of G.

Voronoi separators. We now recall the concept of *Voronoi separators*. A *Voronoi separator* is a sequence of the form

$$S = \langle p_1, u_1, f_1, v_1, p_2, u_2, f_2, v_2, \ldots, p_r, u_r, f_r, v_r \rangle,$$

where p_i are pairwise disjoint objects from $\mathcal{D}$, f_i are faces of G, and u_i, v_i are distinct vertices lying on the face f_i. For each $i \in \{1, \dots, r\}$, define P_i to be the shortest path from u_i to p_i and Q_i to be the shortest path from v_i to p_{i+1}, where indices behave cyclically. For a Voronoi separator S as above, its *length* is r and its *set of traversed objects* is $\mathcal{D}(S) = \{p_1, \dots, p_r\}$.

In the notation above, an object $q \in \mathcal{D}$ is *banned* by the separator S if either q intersects some object $p \in \mathcal{D}(S)$, or there is a vertex w on some path P_i such that $\mathrm{dist}(w, q) < \mathrm{dist}(w, p_i)$, or there is a vertex w on some path Q_i such that $\mathrm{dist}(w, q) < \mathrm{dist}(w, p_{i+1})$. In particular, $\mathcal{D}(S)$ is banned by S. Let $\mathrm{Ban}(S)$ denote the set of objects in $\mathcal{D}$ banned by S. Intuitively, the banned objects are those that are intersected by the separator and are lost when we recurse (in MWISO) or that might be selected and paid twice (in MWDSC). Therefore, we will later ensure that their total weight is small.

The following result is the aforementioned key step toward the proof of Lemma 4. It may be regarded as a lift of Theorem 4.22 from [5] or of Lemma 4.1 from [4] to our setting.

► **Lemma 6.** *Let W be a positive real, $0 < \epsilon < \frac{1}{10}$, and $s = 10^3 \cdot \frac{1}{\epsilon} \ln \frac{1}{\epsilon}$. Suppose $\mathcal{F} \subseteq \mathcal{D}$ is an independent subfamily of objects such that $|\mathcal{F}| \geq 4$, $\mathbf{w}(\mathcal{F}) \leq W$, and $\mathbf{w}(p) \leq s^{-2}W$ for all $p \in \mathcal{F}$. Then there exist a Voronoi separator S satisfying the following:*

(B1) *$\mathcal{D}(S) \subseteq \mathcal{F}$ and all faces traversed by S are $\mathcal{D}$-important;*

(B2) *the length of S is at most $3s$;*

(B3) *the total weight of objects of $\mathcal{F}$ banned by S is at most ϵW;*

(B4) *for every connected component $\mathcal{C}$ of $\mathrm{IntGraph}(\mathcal{D}) - \mathrm{Ban}(S)$, we have $\mathbf{w}(\mathcal{C}) \leq \frac{9}{10}W$.*

It is not hard to see that Lemma 4 follows from Lemma 6: we simply enumerate all candidates for a Voronoi separator S satisfying 1 and 2, a straightforward estimate using Theorem 5 shows that there are at most $6^{3s}N^{15s}$ of them, and for each candidate S we add $\mathrm{Ban}(S)$ to the constructed family $\mathbb{X}$. Details can be found in the full version.

Thus, we are left with proving Lemma 6. The idea, borrowed from Har-Peled [4], is that we construct a sufficiently large random sample from $\mathcal{F}$, where the probability of picking each object is proportional to its weight. Then we inspect the Voronoi diagram induced by the sample and we argue that with non-zero probability it has a short separator giving rise to the sought Voronoi separator S. To implement this plan we need two ingredients: an appropriate lift of the sampling idea from [4] and the analysis of how separators in the Voronoi diagram give rise to Voronoi separators in the graph, which is essentially taken from [5] with some technical details added. These two ingredients are explained in the next two subsections.

2.2 Sampling

Spokes and diamonds. We first adjust technical notions used by Har-Peled [4] in the geometric context to our setting. Suppose we have an independent family of objects $\mathcal{F}$ and its Voronoi diagram $H = H_{\mathcal{F}}$. Let f be any branching point of H and let u be any vertex of f. Let $p \in \mathcal{F}$ be the object of $\mathcal{F}$ such that $u \in \mathbb{M}_{\mathcal{F}}(p)$. The *spoke* of u in H is the shortest path from u to p in G; note all the vertices of this shortest path belong to the cell $\mathbb{M}_{\mathcal{F}}(p)$.

Consider any subfamily $\mathcal{S} \subseteq \mathcal{F}$ and let $H_{\mathcal{S}}$ be the Voronoi diagram induced by $\mathcal{S}$; in the following, we consider spokes in the diagram $H_{\mathcal{S}}$. For any spoke P in $H_{\mathcal{S}}$, say connecting a vertex u with the object $p \in \mathcal{S}$ satisfying $u \in \mathbb{M}_{\mathcal{S}}(p)$, we say that P is *in conflict* with an object $p' \in \mathcal{F}$ if there is a vertex v on P such that $\mathrm{dist}(v, p') < \mathrm{dist}(v, p)$. Note that this implies $p' \notin \mathcal{S}$, because the spoke P has to be entirely contained in $\mathbb{M}_{\mathcal{S}}(p)$. Define the *weight* of P with respect to $\mathcal{F}$ as the total weight of objects from $\mathcal{F}$ that are in conflict with P.

Further, suppose e is an edge of $H_{\mathcal{S}}$, with endpoints f_1, f_2 (not necessarily different). Let p_1, p_2 be the objects of $\mathcal{S}$ corresponding to the faces of $H_{\mathcal{S}}$ incident to e (possibly $p_1 = p_2$).

Let $u_{1,1}, u_{1,2}$ be the vertices of f_1 such that $u_{1,1} \in \mathbb{M}_{\mathcal{S}}(p_1)$, $u_{1,2} \in \mathbb{M}_{\mathcal{S}}(p_2)$, and the edge $u_{1,1}u_{1,2}$ of G crosses the edge e of $H_{\mathcal{S}}$. Similarly pick vertices $u_{2,1}, u_{2,2}$ of f_2. The *diamond* induced by e is the closed curve $\Delta_{\mathcal{S}}(e)$ on Σ formed by the union of: segments connecting the center of f_1 with $u_{1,1}$ and $u_{1,2}$, the unique path between $u_{1,2}$ and $u_{2,2}$ in $\widehat{T}(p_2)$, segments connecting the center of f_2 with $u_{2,2}$ and $u_{2,1}$, and the unique path between $u_{2,1}$ and $u_{1,1}$ in $\widehat{T}(p_1)$. The *interior* of $\Delta_{\mathcal{S}}(e)$ is the unique region of $\Sigma \setminus \Delta_{\mathcal{S}}(e)$ that contains e. The *weight* of $\Delta_{\mathcal{S}}(e)$ with respect to $\mathcal{F}$ is the total weight of objects of $\mathcal{F}$ that are entirely contained in the interior of $\Delta_{\mathcal{S}}(e)$; note that these objects do not belong to $\mathcal{S}$.

We remark that while spokes were used in [4] in the form roughly as above, diamonds correspond to the notion of a *corridor* from [4].

Sampling lemma: statement. We now state the crucial technical result: there is a bounded-size subfamily of the optimum solution that induces a Voronoi diagram where every spoke and every diamond has small weight. We will prove it using a probabilistic sampling argument.

▶ **Lemma 7** (Sampling lemma). *Suppose W is a positive real and $\mathcal{F}$ is an independent, weighted family of objects in G such that $|\mathcal{F}| \geq 4$ and $\mathbf{w}(\mathcal{F}) \leq W$. Let $\ell \geq 10$ be an integer such that $\mathbf{w}(p) \leq \frac{W}{\ell}$ for each $p \in \mathcal{F}$. Then there exists a subfamily $\mathcal{S} \subseteq \mathcal{F}$ with $4 \leq |\mathcal{S}| \leq 2\ell$ such that in the Voronoi diagram $H_{\mathcal{S}}$, the weight with respect to $\mathcal{F}$ of every spoke and of every diamond is at most $10 \ln \ell \cdot \frac{W}{\ell}$.*

Later, we will use Lemma 7 with $\ell = \mathcal{O}((\frac{1}{\epsilon} \ln \frac{1}{\epsilon})^2)$. The reader may imagine that we then apply a balanced planar graph separator on the Voronoi diagram $H_{\mathcal{S}}$ of size $\mathcal{O}(\sqrt{\ell})$ along which we partition $\mathcal{F}$ into two parts, yielding the Voronoi separator claimed by Lemma 6. Since the weight with respect to $\mathcal{F}$ of every spoke and every diamond is at most $10 \ln \ell \cdot \frac{W}{\ell}$, the total weight of the objects of $\mathcal{F}$ banned by S will be bounded by ϵW.

Lemma 7 is a roughly an analogue of Lemma 3.3 from [4]. The main difference is that in the geometric setting, spokes and corridors have a simpler structure due to the fact that each branching point of the Voronoi diagram is defined by three objects from the solution – the three ones equidistant from it – so that the branching point is the meeting point of the three corresponding Voronoi regions. This is no longer the case in planar graphs, as observed in [5]. More precisely, out of the three regions around a branching point of the diagram, two or even three may be equal; this happens when there are bridges in the diagram, which is never the case in the geometric setting.

As part of their proof of Theorem 5, to understand these additional situations Marx and Pilipczuk define *singular faces*, which come in three types. The first one corresponds to "standard" branching points incident to three different regions, while the second and the third one correspond to branching points incident only to two, respectively one region.

Singular faces. For an independent triple of objects $\mathcal{F}_0 = \{p_1, p_2, p_3\} \subseteq \mathcal{D}$, a face f of G is called a *singular face* of *type* 1 for (p_1, p_2, p_3) if in $\mathbb{M}_{\mathcal{F}_0}$, all the vertices of f belong to different cells (note that there are three cells in $\mathbb{M}_{\mathcal{F}_0}$). For an independent triple of objects $\mathcal{F}_0 = \{p_1, p_2, p_3\} \subseteq \mathcal{D}$, a face f is called a *singular face* of *type* 2 for (p_1, p_2, p_3) if in $\mathbb{M}_{\mathcal{F}_0}$, one vertex v_1 of f belongs to $\mathbb{M}_{\mathcal{F}_0}(p_1)$, the other two vertices v_2, v_3 of f belong to $\mathbb{M}_{\mathcal{F}_0}(p_2)$, and the closed walk W obtained by taking the union of the unique path in $\widehat{T}(p_2)$ between v_2 and v_3 and the edge v_2v_3 on the boundary of f divides the plane into two regions, one containing p_1 and one containing p_3. Finally, for an independent quadruple of objects $\mathcal{F}_0 = \{p_0, p_1, p_2, p_3\} \subseteq \mathcal{D}$, a face f is called a *singular face* of *type* 3 for (p_0, p_1, p_2, p_3) if in $\mathbb{M}_{\mathcal{F}_0}$ all the vertices of f belong to $\mathbb{M}_{\mathcal{F}_0}(p_0)$, but the boundary of face f plus the minimal subtree of $\widehat{T}(p_0)$ spanning the vertices of f divides the plane into four regions: the face f itself, one region containing p_1, one region containing p_2, and one region containing p_3. See Figure 8 in [5] for a visualization.

It appears that for a fixed triple or quadruple of objects, there are only few singular faces.

▶ **Lemma 8** (Lemmas 4.8, 4.9, and 4.10 of [5]). *For each independent triple of objects (p_1, p_2, p_3), there are at most 2 singular faces of type 1 for (p_1, p_2, p_3), and at most 1 singular face of type 2 for (p_1, p_2, p_3). For each independent quadruple of objects (p_0, p_1, p_2, p_3), there is at most 1 singular face of type 3 for (p_0, p_1, p_2, p_3).*

The next statement explains the connection between branching points and singular faces.

▶ **Lemma 9** (Lemma 4.12 of [5]). *Let $\mathcal{F} \subseteq \mathcal{D}$ be an independent subfamily of objects, and let H be the Voronoi diagram of $\mathcal{F}$. Then every branching point of H is either a type-1 singular face for some triple of objects from $\mathcal{F}$, or a type-2 singular face for some triple of objects from $\mathcal{F}$, or a type-3 singular face for some quadruple of objects from $\mathcal{F}$.*

Actually, the two results above may be combined into a proof of Theorem 5. Lemma 9 shows that every branching point of the Voronoi diagram of an independent subfamily of $\mathcal{D}$ is among type-1, type-2, and type-3 singular faces for triples or quadruples of objects in $\mathcal{D}$, while using Lemma 8 we can bound their total number by $|\mathcal{D}|^4$.

Sampling lemma: proof. We now have all the tools needed to prove Lemma 7. Contrary to Har-Peled [4] we do not use the Exponential Decay Lemma, but direct probability calculations; this makes the proof somewhat conceptually easier. The main complications are due to the need to handling different types of singular faces, instead of just one.

Proof of Lemma 7. Denote $\eta = 10 \ln \ell \cdot \frac{W}{\ell}$. First observe that if $\mathbf{w}(\mathcal{F}) \le \eta$, then setting $\mathcal{S} = \mathcal{F}$ satisfies all the required properties, since no spoke may have larger weight than the whole of $\mathcal{F}$. Therefore, from now on we assume that $\mathbf{w}(\mathcal{F}) > \eta$.

Construct $\mathcal{S}$ by including every object $p \in \mathcal{F}$ independently with probability $q_p = \mathbf{w}(p) \cdot \frac{\ell}{W}$; note that this value is at most 1 by the assumption of the lemma. Let X be the random variable equal to the cardinality of $\mathcal{S}$; then $X = \sum_{p \in \mathcal{F}} X_p$, where X_p are indicator random variables, taking value 1 if p is included in $\mathcal{F}$ and 0 otherwise. Note that $\mathbb{E}[X_p] = q_p$ and $\mathbb{E}[X] = \sum_{p \in \mathcal{F}} \mathbb{E}[X_p] = \ell \cdot \frac{\mathbf{w}(\mathcal{F})}{W} \le \ell$. Since X is a sum of independent indicator variables, standard concentration inequalities yield the following.

▶ **Claim 10** (♠[4]). *The probability that $|\mathcal{S}| > 2\ell$ or $|\mathcal{S}| \le 100$ is at most $\frac{7}{12}$.*

Call a spoke in the Voronoi diagram $H_{\mathcal{S}}$ *heavy* if its weight with respect to $\mathcal{F}$ is more than η. We now estimate the probability that there is a heavy spoke in $H_{\mathcal{S}}$.

▶ **Claim 11** (♠). *The probability that there is a heavy spoke in $H_{\mathcal{S}}$ is at most $\frac{1}{6}$.*

Proof sketch. Fix any triple of objects $p_1, p_2, p_3 \in \mathcal{F}$ and face f and consider the following event $A^{p_1,p_2,p_3,f}$: p_1, p_2, p_3 are all included in $\mathcal{S}$, f is a branching point of $H_{\mathcal{S}}$ with vertices u_1, u_2, u_3 belonging to the cells of p_1, p_2, p_3, respectively, and moreover the spoke P of u_1 (which is the shortest path from u_1 to p_1) is heavy. Let $\mathcal{Z} \subseteq \mathcal{F}$ be the family of those objects from $\mathcal{F}$ that are in conflict with P; then $\mathbf{w}(\mathcal{Z}) > \eta$. In order for $A^{p_1,p_2,p_3,f}$ to happen, all of

[4] Proofs of claims or lemmas marked with a ♠ are deferred to the full version.

$\{p_1, p_2, p_3\}$ have to be included in $\mathcal{S}$ and none of $\mathcal{Z}$ may be included in $\mathcal{S}$. Since objects are included in $\mathcal{S}$ independently, we have

$$\mathbb{P}(A^{p_1,p_2,p_3,f}) \leq \mathbf{w}(p_1)\mathbf{w}(p_2)\mathbf{w}(p_3) \cdot \frac{\ell^3}{W^3} \cdot \prod_{q \in \mathcal{Z}} (1 - \mathbf{w}(q)) \leq \frac{\mathbf{w}(p_1)\mathbf{w}(p_2)\mathbf{w}(p_3)}{W^3} \cdot \ell^{-7};$$

this follows from simple calculations using the lower bound on $\mathbf{w}(\mathcal{Z})$. By Lemma 8, for each triple $p_1, p_2, p_3 \in \mathcal{F}$ there are at most two singular faces of type 1, hence at most two faces f for which $A^{p_1,p_2,p_3,f}$ has a non-zero probability, each with three vertices. Summing through all the triples $p_1, p_2, p_3 \in \mathcal{F}$, we see that the probability that there is a heavy spoke in $H_\mathcal{S}$ incident to a type-1 branching point is at most $6 \cdot \ell^{-7} \leq 10^{-6}$. Applying a similar reasoning to the other two types of branching points yields the claim. ⌟

We are left with diamonds. Call a diamond $\Delta_\mathcal{S}(e)$ in $H_\mathcal{S}$ *heavy* if its weight with respect to $\mathcal{F}$ is larger than η. The next check follows by essentially the same estimation as Claim 11.

▶ **Claim 12 (♠).** *The probability that there is a heavy diamond in $H_\mathcal{S}$ is at most $\frac{1}{6}$.*

Concluding, assertion $4 \leq |\mathcal{S}| \leq 2\ell$ does not hold with probability at most $\frac{7}{12}$, there is a heavy spoke in $H_\mathcal{S}$ with probability at most $\frac{1}{6}$, and there is a heavy diamond in $H_\mathcal{S}$ with probability at most $\frac{1}{6}$. Hence, with probability at least $\frac{1}{12}$ neither of the above holds, so there exists a subfamily $\mathcal{S}$ satisfying all the postulated conditions. ◀

2.3 Balanced nooses

We proceed with the proof of Lemma 6 by explaining the second ingredient: balanced separators in Voronoi diagrams. In general, short embedding-respecting separators in the Voronoi diagram – so-called *nooses* – correspond to Voronoi separators we are looking for. We start by defining nooses and showing how the existence of a *sphere-cut decomposition* of small width – a hierarchical decomposition of the diagram using nooses – implies the existence of a short noose that breaks the instance in a balanced way.

We remark that in [4], this part of the reasoning is essentially done by considering the *radial* graph of the Voronoi diagram and applying the weighted balanced separator theorem of Miller [6] to it. Such approach would be also applicable in our case, but we find the approach via sphere-cut decompositions more explanatory regarding how separators in the (radial graph of the) diagram correspond to separators in the instance.

Sphere-cut decompositions. We now recall the framework of *sphere-cut decompositions*, which are embedding-respecting hierarchical decompositions of planar graphs.

A *branch decomposition* of a graph H is a pair (T, η) where T is a tree with all internal nodes having degree 3, and η is a bijection between the edge set of H and the leaf set of T (for clarity, we always use the term *node* for a vertex of a decomposition tree). Take any edge e of T and consider removing it from T; then T breaks into two subtrees, say T_1, T_2. Let A_1, A_2 be the preimages of the leaf sets of T_1, T_2 under η, respectively; then (A_1, A_2) is a partition of the edge set of H. The *width* of the edge e is the number of vertices of H incident to both an edge of A_1 and to an edge of A_2, and the *width* of the branch decomposition (T, η) is the maximum among the widths of the edges of T. The *branchwidth* of H is the minimum possible width of a branch decomposition of H.

Let H be a connected plane graph embedded in a sphere Σ. A *noose* in H is a closed, directed curve γ on Σ without self-crossings that meets H only at its vertices and visits every face of H at most once. Note that removing γ from the sphere Σ breaks it into two open disks: for one of them γ is the clockwise traversal of the perimeter, and for the other it is the counter-clockwise traversal (fixing an orientation of Σ). The first disk shall be called $\mathbf{enc}(\gamma)$ while the second shall be called $\mathbf{exc}(\gamma)$ (for *enclosed* and *excluded*). Two nooses γ, γ' are *equivalent* if they are homotopic on Σ with a homotopy that fixes the embedding of H; in other words, γ' can be obtained from γ by continuous transformations within faces of H.

A *sphere-cut decomposition* of H is a triple (T, η, δ) where (T, η) is a branch decomposition of H and δ maps ordered pairs of adjacent nodes of T to nooses on Σ (w.r.t. H) such that the following conditions are satisfied for each pair of adjacent nodes of T:

- $\delta(x, y)$ is equal to $\delta(y, x)$ reversed (in particular $\mathbf{enc}(\delta(x, y)) = \mathbf{exc}(\delta(y, x))$);
- $\mathbf{enc}(\delta(x, y))$ contains all the edges of H mapped to the component of $T - xy$ containing y, while $\mathbf{exc}(\delta(y, x))$ contains all the edges of H mapped to the other component of $T - xy$.

The following result follows from [3, 8] and was formulated in exactly this way in [5].

▶ **Theorem 13.** *Every n-vertex sphere-embedded multigraph that is connected and bridgeless has a sphere-cut decomposition of width at most $\sqrt{4.5n}$.*

We note that in Theorem 13, the assumption that the multigraph is bridgeless is necessary, as multigraphs with bridges do not have sphere-cut decompositions at all.

Suppose (T, η, δ) is a sphere-cut decomposition of G. It is straightforward to see that we may adjust the nooses $\delta(x, y)$ for x, y ranging over adjacent nodes of T so that the following condition is satisfied: if node x has neighbors y_1, y_2, y_3 in T, then $\mathbf{enc}(\delta(y_1, x))$ is the disjoint union of $\mathbf{enc}(\delta(x, y_2))$, $\mathbf{enc}(\delta(x, y_3))$, and $(\delta(x, y_2) \cap \delta(x, y_3)) \setminus \delta(y_1, x)$. Sphere-cut decompositions satisfying this condition will be called *faithful*. It is easy to see that any sphere-cut decomposition can be made faithful by changing each noose to an equivalent one.

Separator theorem for nooses. We now state a separator theorem for nooses drawn from a sphere-cut decomposition of a given sphere-embedded multigraph. The theorem is weighted with respect to a measure defined as follows. Suppose $\mathcal{R}$ is a finite family of pairwise disjoint *objects* on a sphere Σ, where each object $p \in \mathcal{R}$ is a nonempty arc-connected subset of Σ with associated nonnegative weight $\mathbf{w}(p)$. For an open disk $\Delta \subseteq \Sigma$, define its $\mathcal{R}$*-measure* $\mu_{\mathcal{R}}(\Delta)$ as the total weight of objects from $\mathcal{R}$ that are entirely contained in Δ.

▶ **Lemma 14 (♠).** *Let H be a connected, bridgeless multigraph embedded on a sphere Σ. Let $\mathcal{R}$ be a weighted family of pairwise disjoint objects on Σ and let $W = \mathbf{w}(\mathcal{R})$. Suppose further (T, η, δ) is a faithful sphere-cut decomposition of H such that for every pair (x, y) of adjacent nodes in T such that x is a leaf, we have $\mu_{\mathcal{R}}(\mathbf{enc}(\delta(y, x))) < \frac{9}{20}W$. Then there exists a noose γ w.r.t H, which is one of the nooses in the sphere-cut decomposition (T, η, δ), such that the following hold:*

$$\mu_{\mathcal{R}}(\mathbf{enc}(\gamma)) \leq \frac{9}{10}W \qquad \text{and} \qquad \mu_{\mathcal{R}}(\mathbf{exc}(\gamma)) \leq \frac{9}{10}W.$$

The proof of Lemma 14 is standard: we find a balanced edge (x, y) in the decomposition (T, η, δ) and $\delta(x, y)$ is the sought noose. The fact that nooses appearing in (T, η, δ) may intersect objects from $\mathcal{R}$ requires some technical attention. Details are in the full version.

Proof sketch of Lemma 6. The proof of Lemma 6 now essentially follows by combining Theorem 13, Lemma 7, and Lemma 14 as follows. Applying Lemma 7 to $\mathcal{F}$ with $\ell = s^2$ yields a suitable subfamily $\mathcal{S} \subseteq \mathcal{F}$. We investigate the Voronoi diagram $H_{\mathcal{S}}$ induced by $\mathcal{S}$. Applying Theorem 13 to $H_{\mathcal{S}}$ yields a sphere-cut decomposition of width at most $3s$, which we can feed to Lemma 14 to obtain a balanced noose γ. This noose naturally corresponds to a Voronoi separator $S(\gamma)$, obtained by essentially tracing γ and marking the parts of the Voronoi diagram of $\mathcal{S}$ that it visits. The connection between nooses in the Voronoi diagram and Voronoi separators was largely explored in [5]. Then properties asserted by Lemma 7, in particular the fact that every spoke in $H_{\mathcal{S}}$ has small weight, imply that $S(\gamma)$ satisfies all the requested conditions. There is one technical caveat in that the Voronoi diagram $H_{\mathcal{S}}$ may have bridges, so a priori we cannot apply Theorem 13 to it; this requires special technical treatment. The detailed proof of Lemma 6 can be found in the full version.

3 A QPTAS for Maximum Weight Independent Set of Objects

In this section we use Lemma 4 to design a QPTAS for MWISO, that is, we prove Theorem 1. Recall that the setting is as follows. The input is $(G, \mathcal{D})$, where G is a graph embedded in a sphere Σ together with a family of objects $\mathcal{D}$, each being a connected subgraph of G with a prescribed positive weight. Moreover, we are given an accuracy parameter $\epsilon > 0$ and we may assume w.l.o.g. that $\epsilon < \frac{1}{10}$. The goal is to find an independent subfamily $\mathcal{F} \subseteq \mathcal{D}$ with the largest possible weight; more precisely, the algorithm shall compute a solution of weight at least $(1-\epsilon)$ times the optimum. Let $n = |V(G)|$ and $N = |\mathcal{D}|$; w.l.o.g. we assume $N \geq 2$.

We will also assume that all objects in $\mathcal{D}$ have weights between 1 and $M = 2\epsilon^{-1}N$. This assumption is easy to achieve as follows: guess the heaviest object p from the optimum solution, remove all objects of weight less $\mathbf{w}(p)/M$ from $\mathcal{D}$, rescale the weights to the interval $[1, M]$, and then look for an $(1-\epsilon/2)$-approximate solution. To see that this is correct, observe that in the optimum solution, objects of weight less than $\mathbf{w}(p)/M$ in total constitute at most an $\epsilon/2$-fraction of the weight of p alone, so by removing them we lose at most an $\epsilon/2$ fraction of the optimum. A formal reasoning is presented in the full version.

Before we proceed to the algorithm, we fix the following parameters:

$$d_{\max} = 10\ln(MN), \qquad \hat{\epsilon} = \frac{\epsilon}{d_{\max}}, \qquad s = 10^3 \cdot \frac{1}{\hat{\epsilon}} \ln \frac{1}{\hat{\epsilon}}.$$

Parameter $d_{\max}$ is the maximum recursion depth of the algorithm; note that $d_{\max} = \mathcal{O}(\log(N/\epsilon))$. Next, $\hat{\epsilon} = \mathcal{O}(\epsilon/\log(N))$ is the refined accuracy parameter which will be used throughout the recursion instead of ϵ. Similarly as in [1, 2, 4], intuitively we lose a factor of $1-\hat{\epsilon}$ in each recursion level which yields an overall approximation ratio of $(1-\hat{\epsilon})^{d_{\max}} = (1-\epsilon/\log(N))^{\mathcal{O}(\log(N))} = 1 - \mathcal{O}(\epsilon)$. Let us stress that although the algorithm uses recursion and the number of objects changes in subsequent recursive calls, the values of $d_{\max}, \hat{\epsilon}, s$ are fixed as above and their definitions always refer to the initial number of objects.

We now explain the algorithm; it is also summarized using pseudocode as Algorithm 1 Let us fix an optimum solution $\mathcal{F}_{\mathrm{OPT}}$ and denote $W = \mathbf{w}(\mathcal{F}_{\mathrm{OPT}})$. We shall analyze $\mathcal{F}_{\mathrm{OPT}}$, which will lead to the formulation of the algorithm as a recursive search for $\mathcal{F}_{\mathrm{OPT}}$.

We would like to use Lemma 4 to guess a Voronoi separator that breaks $\mathcal{F}_{\mathrm{OPT}}$ in a balanced way. However, we first need make sure that every object in question constitutes only a small fraction of W. This is done by a standard method of guessing exactly "heavy" objects in the solution, whose number is small, and proceeding only with the "light" ones.

More precisely, call an object $p \in \mathcal{F}_{\mathrm{OPT}}$ *heavy* if $\mathbf{w}(p) > s^{-2}W$. Observe that the number of heavy objects in $\mathcal{F}_{\mathrm{OPT}}$ is at most s^2, hence there are at most N^{s^2} possible ways to select those heavy objects from $\mathcal{D}$. The algorithm branches into all possible such ways, in each branch fixing a different candidate for the set of heavy objects. Hence, by increasing the number of subproblems by a multiplicative factor N^{s^2} we may assume that the algorithm fixes the set $\mathcal{F}_{\mathrm{hv}}$ consisting of all heavy objects in $\mathcal{F}_{\mathrm{OPT}}$. Let $\mathcal{D}'$ be obtained from $\mathcal{D}$ by

Algorithm 1: Algorithm `AlgMWISO`.

```
Input: An instance $(G, \mathcal{D})$, recursion depth $d$
Output: An independent family $\mathcal{F} \subseteq \mathcal{D}$

  if $d > d_{\max}$ then
      return $\emptyset$
  $\mathcal{F} \leftarrow \emptyset$
  forall $\mathcal{F}_{\text{hv}}$: independent subfamily of $\mathcal{D}$ with $|\mathcal{F}_{\text{hv}}| \leq s^2$ do
      $\mathcal{D}' \leftarrow \mathcal{D} -$ (all objects that intersect any object of $\mathcal{F}_{\text{hv}}$)
      $\mathbb{X} \leftarrow$ family computed for $\mathcal{D}'$ using Lemma 4
      forall $\mathcal{X} \in \mathbb{X}$ do
          $\mathcal{D}_1, \ldots, \mathcal{D}_k \leftarrow$ vertex sets of the connected components of
           $\text{IntGraph}(\mathcal{D}') - \mathcal{X}$
          for $i = 1$ to $k$ do
              $\mathcal{F}_i \leftarrow$ AlgMWISO$(G, \mathcal{D}_i, d+1)$
          $\mathcal{F}_{\text{cand}} \leftarrow \mathcal{F}_{\text{hv}} \cup \bigcup_{i=1}^{k} \mathcal{F}_i$
          if $\mathbf{w}(\mathcal{F}_{\text{cand}}) > \mathbf{w}(\mathcal{F})$ then
              $\mathcal{F} \leftarrow \mathcal{F}_{\text{cand}}$
  return $\mathcal{F}$
```

removing $\mathcal{F}_{\text{hv}}$ and all objects intersecting any object from $\mathcal{F}_{\text{hv}}$, and let $\mathcal{F}'_{\text{OPT}} = \mathcal{F}_{\text{OPT}} - \mathcal{F}_{\text{hv}}$. Note that $\mathbf{w}(\mathcal{F}'_{\text{OPT}}) \leq W$ and $\mathbf{w}(p) \leq s^{-2}W$ for all $p \in \mathcal{F}'_{\text{OPT}}$.

We may now apply Lemma 4 to $\mathcal{F}'_{\text{OPT}} \subseteq \mathcal{D}'$ with W being the upper bound on its weight. Thus, in time $N^{\mathcal{O}(s)} \cdot n^{\mathcal{O}(1)}$ we may compute a family $\mathbb{X}$ consisting of subsets of $\mathcal{D}'$ with $|\mathbb{X}| \leq 6^{3s}N^{15s}$ and satisfying the following property: there exists $\mathcal{X} \in \mathbb{X}$ such that $\mathbf{w}(\mathcal{F}'_{\text{OPT}} \setminus \mathcal{X}) \leq \epsilon W$ and within each connected component of the graph $\text{IntGraph}(\mathcal{D}') - \mathcal{X}$ the total weight of objects from $\mathcal{F}'_{\text{OPT}}$ does not exceed $\frac{9}{10}W$. By branching into all the members of $\mathbb{X}$, via increasing the number of subproblems by a multiplicative factor $|\mathbb{X}|$ we may henceforth assume that the algorithm fixes $\mathcal{X}$ with properties as above.

For a fixed choice of $\mathcal{F}_{\text{hv}}$ and $\mathcal{X}$ as above, let us inspect the connected components of $\text{IntGraph}(\mathcal{D}') - \mathcal{X}$; let their vertex sets be $\mathcal{D}_1, \ldots, \mathcal{D}_k$. We apply the algorithm recursively to the instances $(G, \mathcal{D}_i)$ for $i = 1, \ldots, k$, yielding independent families $\mathcal{F}_1, \ldots, \mathcal{F}_k$ with $\mathcal{F}_i \subseteq \mathcal{D}_i$. We record the family $\mathcal{F} = \mathcal{F}_{\text{hv}} \cup \bigcup_{i=1}^{k} \mathcal{F}_i$ as a candidate for the solution; it is straightforward to see from the construction that this family is independent. Finally, as the final solution we output the heaviest among the recorded candidates; that is, the heaviest solution found for all choices of $\mathcal{F}_{\text{hv}}$ and $\mathcal{X}$. We remark that if for some choice of $\mathcal{F}_{\text{hv}}$ it turned out that $\mathcal{D}' = \emptyset$, i.e., every object intersects some objects from $\mathcal{F}_{\text{hv}}$, then $\mathbb{X}$ contains only one choice of $\mathcal{X}$ being $\emptyset$, hence we include $\mathcal{F} = \mathcal{F}_{\text{hv}}$ among the candidates without invoking any recursive calls.

The base case of the recursion is provided by trimming it at level $d_{\max}$. More precisely, all subcalls at depth larger than $d_{\max}$ return empty solutions. This concludes the description of the algorithm; as mentioned, it is summarized using pseudocode as Algorithm 1.

The above algorithm runs in time $2^{\text{poly}(1/\epsilon, N)} \cdot n^{\mathcal{O}(1)}$, because at each node of the recursion tree the algorithm uses polynomial time and calls itself on $N^{\mathcal{O}(s^2)}$ subinstances, where $s = \text{poly}(1/\epsilon, \log N)$. Together with the bound of $d_{\max} = \mathcal{O}(\log(N/\epsilon))$ on the recursion depth this yields the promised running time. As mentioned, the claimed approximation ratio follows as intuitively in each of the $d_{\max}$ recursion levels we lose a factor of $1 - \hat{\epsilon}$, accumulating to $(1 - \hat{\epsilon})^{d_{\max}} = 1 - \mathcal{O}(\epsilon)$ overall. Formal proofs are in the full version.

References

1 Anna Adamaszek and Andreas Wiese. Approximation schemes for maximum weight independent set of rectangles. In *Proc. FOCS 2013*, pages 400–409. IEEE, 2013.

2 Anna Adamaszek and Andreas Wiese. A QPTAS for maximum weight independent set of polygons with polylogarithmically many vertices. In *Proc. SODA 2014*, pages 645–656. SIAM, 2014.

3 Qian-Ping Gu and Hisao Tamaki. Improved bounds on the planar branchwidth with respect to the largest grid minor size. *Algorithmica*, 64(3):416–453, 2012.

4 Sariel Har-Peled. Quasi-polynomial time approximation scheme for sparse subsets of polygons. In *Proc. SoCG 2014*, pages 120–129. SIAM, 2014.

5 Dániel Marx and Michał Pilipczuk. Optimal parameterized algorithms for planar facility location problems using Voronoi diagrams. In *Proc. ESA 2015*, volume 9294 of *LNCS*, pages 865–877. Springer, 2015.

6 Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. Syst. Sci.*, 32(3):265–279, 1986.

7 Nabil H. Mustafa, Rajiv Raman, and Saurabh Ray. Quasi-polynomial time approximation scheme for weighted geometric set cover on pseudodisks and halfspaces. *SIAM J. Comput.*, 44(6):1650–1669, 2015.

8 Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

On Learning Linear Functions from Subset and Its Applications in Quantum Computing

Gábor Ivanyos
Institute for Computer Science and Control, Hungarian Academy of Sciences,
Budapest, Hungary
Gabor.Ivanyos@sztaki.mta.hu

Anupam Prakash
CNRS, IRIF, Université Paris Diderot 75205 Paris, France
anupam@irif.fr

Miklos Santha
CNRS, IRIF, Université Paris Diderot 75205 Paris, France; and
Centre for Quantum Technologies, National University of Singapore, Singapore 117543
miklos.santha@gmail.com

Abstract

Let $\mathbb{F}_q$ be the finite field of size q and let $\ell : \mathbb{F}_q^n \to \mathbb{F}_q$ be a linear function. We introduce the *Learning From Subset* problem $\mathrm{LFS}(q,n,d)$ of learning ℓ, given samples $u \in \mathbb{F}_q^n$ from a special distribution depending on ℓ: the probability of sampling u is a function of $\ell(u)$ and is non zero for at most d values of $\ell(u)$. We provide a randomized algorithm for $\mathrm{LFS}(q,n,d)$ with sample complexity $(n+d)^{O(d)}$ and running time polynomial in $\log q$ and $(n+d)^{O(d)}$. Our algorithm generalizes and improves upon previous results [8, 10] that had provided algorithms for $\mathrm{LFS}(q,n,q-1)$ with running time $(n+q)^{O(q)}$. We further present applications of our result to the *Hidden Multiple Shift* problem $\mathrm{HMS}(q,n,r)$ in quantum computation where the goal is to determine the hidden shift s given oracle access to r shifted copies of an injective function $f : \mathbb{Z}_q^n \to \{0,1\}^l$, that is we can make queries of the form $f_s(x,h) = f(x-hs)$ where h can assume r possible values. We reduce $\mathrm{HMS}(q,n,r)$ to $\mathrm{LFS}(q,n,q-r+1)$ to obtain a polynomial time algorithm for $\mathrm{HMS}(q,n,r)$ when $q = n^{O(1)}$ is prime and $q-r = O(1)$. The best known algorithms [5, 8] for $\mathrm{HMS}(q,n,r)$ with these parameters require exponential time.

2012 ACM Subject Classification Theory of computation → Quantum computation theory

Keywords and phrases Learning from subset, hidden shift problem, quantum algorithms, linearization

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.66

Related Version A full version of the paper is available at [11], `https://arxiv.org/abs/1710.02581`.

Funding A part of the research was accomplished while the first two authors were visiting the Centre for Quantum Technologies (CQT), National University of Singapore. The research at CQT was partially funded by the Singapore Ministry of Education and the National Research Foundation under grant R-710-000-012-135. This research was supported in part by the QuantERA ERA-NET Cofund project QuantAlgo and by the Hungarian National Research, Development and Innovation Office – NKFIH, Grant K115288.

© Gábor Ivanyos, Anupam Prakash, and Miklos Santha;
licensed under Creative Commons License CC-BY

1 Introduction

1.1 Learning with noise

Let $n \geq 1$ and $q > 1$ be integers. We denote by $\mathbb{Z}_q$ the ring of integers modulo q, and by $\mathbb{F}_q$ the finite field on q elements, when q is some power of a prime number. When q is prime then $\mathbb{Z}_q$ coincides with $\mathbb{F}_q$, and we will use the notation $\mathbb{F}_q$. Let $\ell : \mathbb{F}_q^n \to \mathbb{F}_q$ be an n-variable linear function. The main subject of this paper is to learn ℓ given partial information about the values $\ell(u)$ for uniformly random samples u from $\mathbb{F}_q^n$. In the ideal setting, when we have access to the values $\ell(u)$ for uniformly random samples from $\mathbb{F}_q^n$, the problem is canonical and perfectly understood: after getting n independent samples, we can determine ℓ by Gaussian elimination in polynomial time. But when instead of the exact values we receive only some property satisfied by them, the problem can become much more difficult.

Since an element of $\mathbb{F}_q^n$ can be specified with $n \log q$ bits, we will say that *an algorithm is in polynomial time* if it runs in time polynomial in both n and $\log q$. Let $f(n, q)$ be a function of n and q, then we say that a function $g(n, q) \in \widetilde{O}(f)$ if $g(n, q) \leq f(n, q) \log^c(nq)$ for some constant c for sufficiently large n and q. By the *sample complexity* of an algorithm we mean the number of samples used by it.

There is a somewhat similar context to the learning model we investigate, it is the model where the values $\ell(u)$ are perturbed by some random noise. The first example of such a work is by Blum et al. [3] on the *Learning Parity with Noise* problem $\mathrm{LPN}(n, \eta)$, where $\eta < 1/2$. Here we have access to tuples $(u, b) \in \mathbb{F}_2^n \times \mathbb{F}_2$, where u is a uniformly random element of $\mathbb{F}_2^n$ and $b = \ell(u) + e$, where e is a random 0–1 variable with $\Pr[e = 1] = \eta$. For constant noise rate $0 < \eta < 1/2$, the best known algorithm for $\mathrm{LPN}(n, \eta)$ is from [3]. It has both sample and time complexity of $2^{O(n/\log n)}$, and therefore only marginally beats the trivial exhaustive search algorithm of complexity $2^{O(n)}$.

The *Learning With Error* problem $\mathrm{LWE}(q, n, \chi)$ is a generalization by Regev [17] of LPN to larger fields. Here q can be any prime number, and χ is a probability distribution on $\mathbb{F}_q$. Similar to LPN, we have access to tuples $(u, b) \in \mathbb{F}_q^n \times \mathbb{F}_q$, where u is a uniformly random element of $\mathbb{F}_q^n$ and $b = \ell(u) + e$, with the random variable e having distribution χ. Under the assumptions that q is bounded by some polynomial function of n, and that $\chi(0) \geq 1/q + 1/p(n)$, for some polynomial p, the problem can be solved classically with sample and time complexity $2^{O(n)}$. The case when $\chi = \Psi_\alpha$, the discrete Gaussian distribution of standard deviation αq, is of particular interest for lattice based cryptography. Indeed, one of the main results of [17] is that for appropriate parameters, solving $\mathrm{LWE}(q, n, \Psi_\alpha)$ is at least as hard as quantumly solving several cryptographically important lattice problems in the worst case. In a subsequent work a classical reduction of some of these lattice problems to LWE was given by Peikert [15].

In [2] Arora and Ge introduced a more structured noise model for learning linear functions over $\mathbb{F}_2^n$. In the *Learning Parity with Structured Noise* problem $\mathrm{LPSN}(n, m)$ the samples arrive in groups of size m, that is in one sampling step we receive $(u_1, b_1), \ldots, (u_m, b_m)$, where $(u_i, b_i) \in \mathbb{F}_2^n \times \mathbb{F}_2$, for $i = 1, \ldots, m$. Here $u_1, \ldots, u_m$ are independent random elements drawn from $\mathbb{F}_2^n$, and $b_i = \ell(u_i) + e_i$, where the the noise vector $e = (e_1, \ldots, e_m) \in \mathbb{F}_2^m$ must have Hamming weight less than $m/2$. The chosen noise vector e can depend on the sample $(u_1, \ldots, u_m)$, but the model has an important restriction (structure) compared to the previous error models. Since the Hamming weight of e is less than $m/2$, it is guaranteed that in every sampling group the majority of the bits b_i is correct, that is coincides with $\ell(u_i)$. In fact, the model of Arora and Ge is somewhat more general. Let P be any m-variable polynomial over $\mathbb{F}_2^m$, for which there exists $a \in \mathbb{F}_2^m$, such that $a \neq c + c'$ for all $c, c' \in \mathbb{F}_2^m$

satisfying $P(c) = P(c') = 0$. Then the error vector can be any $e \in \mathbb{F}_2^m$ satisfying $P(e) = 0$. The main result of [2] is that $\mathrm{LPSN}(n, m)$ can be solved in time $n^{O(m)}$, implying that the linear function can be learnt in polynomial time when m is constant.

1.2 Learning from subset

We consider here a different model of learning linear functions where the difficulty doesn't come from the noisy sampling process, but from the fact that instead of obtaining the actual values of the sampled elements, we only receive some partial information about them.

Such a model was first considered by Friedl et al. [8] with the *Learning From Disequations* problem $\mathrm{LFD}(q, n)$ where q is a prime number. Here we never get sample elements from the kernel of ℓ, that is we can only sample u if $\ell(u) \neq 0$, which explains the name of the problem. Friedl et al. [8] consider distributions p which are not necessarily uniform on their support, in fact they only require that $p(u) = p(v)$ whenever $\ell(u) = \ell(v)$.

The reason to consider this learning problem in [8] is that the *Hidden Shift* problem $\mathrm{HS}(q, n)$, a paradigmatic problem in quantum computing, can be reduced in quantum polynomial time to $\mathrm{LFD}(q, n)$. In $\mathrm{HS}(q, n)$ we have oracle access to two injective functions f_0 and f_1 over $\mathbb{F}_q^n$ with the promise that for some element $s \in \mathbb{F}_q^n$, we have $f_1(x) = f_0(x - s)$, for all $x \in \mathbb{F}_q^n$. The element s is called the *hidden shift*, and the task is to find it. It is proven in [8] that $\mathrm{LFD}(q, n)$ can be solved in time $(n+q)^{O(q)}$. This result implies that there exists a quantum algorithm for $\mathrm{HS}(q, n)$ of similar complexity. When q is constant, these algorithms are therefore polynomial time.

In a subsequent paper [10] Ivanyos extended the work of [8] to the case when q is a prime power, both for $\mathrm{LFD}(q, n)$ and $\mathrm{HS}(q, n)$. The complexity bounds obtained are very similar to the bounds of [8], and therefore his results imply that $\mathrm{LFD}(q, n)$ can be solved in polynomial time, and that $\mathrm{HS}(q, n)$ in quantum polynomial time when q is a prime power of constant size.

Observe that the complexity bound $(n + q)^{O(q)}$ is not only not polynomial in $\log q$, but is not even exponential, in fact it is doubly exponential. Therefore [8] and [10] not only leave open the question whether, in general, it is possible to obtain a (quantum) algorithm for $\mathrm{LFD}(q, n)$ and $\mathrm{HS}(q, n)$ with running time polynomial in n and q, but also the question of the existence of algorithms which have running time polynomial in n and $\log q$. These questions are still open today.

In this work we introduce a generalization of the learning problem LFD. While in LFD the sampling distribution had to avoid the kernel of ℓ, in our model the input contains a set $A \subseteq \mathbb{F}_q$, and we sample from distributions whose support contains only those elements u, for which $\ell(u) \in A$. As in [8], we require that the elements with the same ℓ-value have identical probabilities. We allow these probabilities to be exponentially small and even 0.

▶ **Definition 1.** Let $A \subset \mathbb{F}_q$, where q is a prime power, let $\ell : \mathbb{F}_q^n \to \mathbb{F}_q$ be a linear function, and let p be a distribution over $\mathbb{F}_q^n$. We say that the *ℓ-image* of p is A if $\ell(\mathrm{supp}(p)) = A$. The distribution is *$\ell$-symmetric* if $\ell(u) = \ell(v)$ implies $p(u) = p(v)$. If the ℓ-image of p is a subset of A and p is also ℓ-symmetric, we say that p is an *(A, ℓ)-distribution*.

In other words, p is an (A, ℓ)-distribution if p is constant on each affine subspace $V_\alpha = \{u \in \mathbb{F}_q^n : \ell(u) = \alpha\}$, for $\alpha \in \mathbb{F}_q$, and moreover p is zero on V_α, whenever $\alpha \notin A$. It is not hard to see that for $|A| < q$, if p is simultaneously an (A, ℓ)-distribution and an (A, ℓ')-distribution then ℓ' is a constant multiple of ℓ. On the other hand, non-zero constant multiples of a linear function can not be distinguished in general in this model: for example, if $A = \mathbb{F}_q \setminus \{0\}$, then for every $c \neq 0$, an (A, ℓ)-distribution is also an $(A, c\ell)$-distribution.

▶ **Definition 2.** The *Learning From Subset* problem $\mathrm{LFS}(q,n,d)$ is parametrized by three positive integers q,n and d, where q is a prime power and $2 \le d \le q-1$.
Input: A set $A \subset \mathbb{F}_q$ of cardinality d and a sequence of N samples $u_1,\ldots,u_N$ from an (A,ℓ)-distribution for some nonzero linear function $\ell : \mathbb{F}_q^n \to \mathbb{F}_q$.
Output: A non zero constant multiple of ℓ.

For $d < d'$, an $\mathrm{LFS}(q,n,d)$ instance is also an $\mathrm{LFS}(q,n,d')$ instance, therefore the problem is harder for bigger d. For $d=1$ the problem is simple because it becomes a system of linear equalities which can be solved by Gaussian elimination. When $d=q$ we don't receive any information from the samples and it is impossible to identify the linear function. When $d=q-1$ and $A=\mathbb{F}_q \setminus \{0\}$, the problem LFS specializes to LFD, in fact the latter is the hardest instance of the former.

The first main result of our paper is a randomized algorithm for $\mathrm{LFS}(q,n,d)$ whose complexity depends exponentially on d, but only polynomially on $\log q$. This result shows that the increase of information by reducing the size of the set A can indeed be algorithmically exploited. More precisely, we show that for a sample size N which is a sufficiently large polynomial of n^d, there exists a randomized algorithm which in time polynomial in n^d and $\log q$, with probability $1/2$, determines ℓ up to a constant factor.

▶ **Theorem 3.** *There is a randomized algorithm for* $\mathrm{LFS}(q,n,d)$ *with sample complexity* $(n+d)^{O(d)}$ *and running time polynomial in* $\log q$ *and* $(n+d)^{O(d)}$.

The main interest of this result is that for constant d it gives a polynomial time algorithm for LFS. For $d=q-1$ and $A=\mathbb{F}_q \setminus \{0\}$ it yields the same complexity bound as [8] and [10]. But observe, that even for non constant $d=o(q)$, it is asymptotically faster than the algorithms in the above papers.

1.3 Hidden multiple shifts

The original motivation for [8] to study LFD was its connection to the hidden shift problem. This problem was implicitly introduced by Ettinger and Høyer [7], while studying the dihedral hidden subgroup problem. The hidden shift problem can be defined in any group G. We are given two injective functions f_0 and f_1 mapping G to some arbitrary finite set. We are promised that for some element $s \in G$, we have $f_1(xs) = f_0(x)$, for every $x \in G$, and the task is to find s. As shown in [7], when G is abelian, the hidden shift in G is quantum polynomial time equivalent to the hidden subgroup problem in the semidirect product $G \rtimes \mathbb{Z}_2$. In the semidirect product the group operation is defined as $(x_1,b_1).(x_2,b_2) = (x_1 + (-1)^{b_1}x_2, b_1+b_2)$, and the function $f(x,b) = f_b(x)$ hides the the subgroup $\{(0,0),(s,1)\}$. The quantum complexity of HS in the cyclic group $\mathbb{Z}_q$ (or equivalently, the complexity of the hidden subgroup in the dihedral group $\mathbb{Z}_q \rtimes \mathbb{Z}_2$) is a famous open problem in quantum computing. In [7] there is a quantum algorithm for this problem of polynomial quantum sampling complexity, but followed by an exponential time classical post-processing. The currently best known quantum algorithm is due to Kuperberg [13], and it is of subexponential complexity $2^{O(\sqrt{\log q})}$. Note that one could also consider shifts of non-injective functions. The extension of HS to such cases can become quite difficult even over $\mathbb{Z}_2^n$ where HS for injective functions is identical to the hidden subgroup problem. Results in this direction can be found e.g. in [9], [4] and [18].

As one could expect, the polynomial time algorithm for LFS with constant d has further consequences for quantum computing. Indeed, using this learning algorithm, we can solve in quantum polynomial time some instances of the hidden multiple shifts problem, a generalization of the hidden shift problem, which we define now.

For an element $s \in \mathbb{Z}_q^n$, a subset $H \subseteq \mathbb{Z}_q$ of cardinality at least 2, and a function $f : \mathbb{Z}_q^n \to \{0,1\}^l$, where l is an arbitrary positive integer, we define the function $f_s : \mathbb{Z}_q^n \times H \mapsto \{0,1\}^l$ as $f_s(x,h) = f(x - hs)$. We think about $f_s(x,h)$ as the hth *shift* of f by s. The task in the hidden multiple shift problem is to recover s when we are given oracle access, for some f and H, to f_s. This problem doesn't necessarily have a unique solution. Indeed, let us define $\delta(H,q)$ as the largest divisor of q such that $h - h'$ is divisible by $\delta(H,q)$ for every $h, h' \in H$. Pick $h_0 \in H$. Then for any $s' \in \frac{q}{\delta(H,q)}\mathbb{Z}_q^n$ and $h \in H$, we have $hs' = h_0 s' + (h - h_0)s' = h_0 s'$ whence $h(s + s') = hs + h_0 s'$ and therefore

$$f_{s+s'}(v,h) = f(v - h(s+s')) = f(v - h_0 s' - hs) = f'_s(v,h),$$

where $f'(v) = f(v - h_0 s')$. This means that s and $s + s'$ are indistinguishable by the set of shifts of f, and therefore we can only hope to determine (the coordinates of) s modulo $\frac{q}{\delta(H,q)}$. When q is a prime number, this problem of course doesn't arise.

▶ **Definition 4.** The *Hidden Multiple Shift* problem $\mathrm{HMS}(q,n,r)$ parametrized by three positive integers q, n and r, where $q > 1$ and $2 \le r \le q - 1$.
Input: A set $H \subseteq \mathbb{Z}_q$ of cardinality r.
Oracle input: A function $f_s : \mathbb{Z}_q^n \times H \to \{0,1\}^l$, where $s \in \mathbb{Z}_q^n$ and $f : \mathbb{Z}_q^n \to \{0,1\}^l$ is an injective function.
Output: $s \mod \frac{q}{\delta(H,q)}$.

The HMS problem was first considered by Childs and van Dam [5]. They investigated the cyclic case $n = 1$ and assumed that H is a contiguous interval and presented a polynomial time quantum algorithm for such an H of size $q^{\Omega(1)}$. Their result could probably be extended to constant n. However, for 'medium-size' n and q, such a result seems to be very difficult to achieve. Obtaining an efficient algorithm for medium sized n, q is also stated as an open problem [6], and it is noted that such a result would greatly simplify their algorithm. Intuitively, for small H the HMS appears to be 'too close' to the HS for which the so far best result is still what is given in [8].

For $r = q$, the HMS problem can be solved in quantum polynomial time. Indeed, in that case $H = \mathbb{Z}_q$, and $\mathbb{Z}_q^n \times H = \mathbb{Z}_q^{n+1}$ is an abelian group. The function f_s hides the subgroup generated by $(s,1)$, therefore we have an instance of the abelian hidden subgroup problem. When $r = 1$ the problem is void, there is no hidden shift. When $r = 2$, we have the standard hidden shift problem for which [8] and [10] gave a quantum algorithm of complexity $(n+q)^{O(q)} = (n+q)^{O(q+1-r)}$. Their method at a high level is a quantum reduction to (several instances of) $\mathrm{LFS}(q,n,q-1)$. These extreme cases suggest a strong connection between the classical complexity of $\mathrm{LFS}(q,n,d)$ and the quantum complexity of $\mathrm{HMS}(q,n,r)$ when $r = q + 1 - d$. Indeed, this turns out to be true. In our second main result we give a polynomial time quantum Turing reduction of $\mathrm{HMS}(q,n,r)$ to $\mathrm{LFS}(q,n,q+1-r)$, to obtain an algorithm of complexity $(n+q)^{O((q-r)^2)}$ for the former problem.

▶ **Theorem 5.** *Let q be a prime. Then there is a quantum algorithm which solves* $\mathrm{HMS}(q,n,r)$ *with sample complexity and in time* $(n+q)^{O((q-r)^2)}$.

The above Theorem yields a polynomial time algorithm for $\mathrm{HMS}(q,n,r)$ for the case when $q - r$ is constant and $q = n^{O(1)}$. We also present a Fourier sampling based algorithm for HMS which is polynomial time for a different set of parameters satisfying $\frac{r}{q} = 1 - \Omega(\frac{\log n}{n})$. We have the following result.

▶ **Theorem 6.** *There is a quantum algorithm that solves* $\mathrm{HMS}(q,n,r)$ *with high probability in time* $O(\mathrm{poly}(n)(\frac{q}{r})^{n+O(1)})$.

1.4 Our proof methods

The basic idea of the proof of Theorem 3 is a variant of linearization used in [8] and in [2], presented in the flavor of [10]. To give a high level description, observe that every u such that $p(u) \neq 0$ is a zero of the polynomial $f_{(A,\ell)}(x) = \prod_{a \in A}(\ell(x) - a)$. By Hilbert's Nullstellensatz, over the algebraic closure of $\mathbb{F}_q$, the polynomials which vanish on all the zeros of $f_{(A,\ell)}$ are multiples of $f_{(A,\ell)}$. In particular, every such polynomial which is also of degree at most d must be a scalar multiple of $f_{(A,\ell)}$. Interestingly, one could show that this consequence remains true with high probability, if we replace "all the zeros" by sufficiently many random samples provided that our (A, ℓ)-distribution is uniform (or nearly uniform) in the sense that $p(u)$ (the probability of sampling u) is the same (or almost the same) for every u such that $\ell(u) \in A$, independently on the actual value of $\ell(u)$. Therefore, in the (nearly) uniform case one could compute a nontrivial scalar multiple of $f_{(A,\ell)}$ by finding a nontrivial solution of a system of N homogeneous linear equations in $(n + d)^d$ unknowns (these are the coefficients of the various monomials in $f_{(A,\ell)}$). Then ℓ could be determined by factoring this polynomial. This method would be a direct generalization of the algorithms given in [8] and [10]. Indeed, in those papers one could just take $A = \mathbb{F}_q \setminus \{0\}$. However, the proofs (and in case of [10] even the algorithmic ingredients) are designed specially for small q and straightforward extensions would result in algorithms of complexity depending exponentially not only on d but on $\log q$ as well. Here we give an algorithm that depends polynomially on $\log q$ and that works without any assumption on uniformity. (In the case $A = \mathbb{F}_q \setminus \{0\}$ uniformity can actually be simulated by multiplying the sample vectors by random nonzero scalars.) Then, instead of divisibility by $f_{(A,\ell)}$ we prove that, with high probability, the polynomials that are zero on sufficiently many samples are divisible by $\ell(x) - a$ for the "most frequent" value $a \in A$. Then we find a scalar multiple of ℓ by factoring a nonzero polynomial from the space of those which are zeros on all the samples.

The subexponential LWE-algorithm of Arora and Ge [2] is based on implicitly solving a problem that can be cast as an instance of LFS where one of the coefficients of the linear function ℓ is known, $0 \in A$, and the (A, ℓ) distribution is such that 0 is the most likely value. More details are given in the full version [11].

The algorithm for solving $\mathrm{HMS}(q, n, r)$ in Theorem 5 is based on the following. After applying some standard preprocessing, we obtain samples of states that are projections to an r-dimensional space of $\mathrm{QFT}(|(u, s)\rangle)$ where QFT denotes the quantum Fourier transform on $\mathbb{Z}_q^n$, the vector $u \in \mathbb{Z}_q^n$ is sampled from the uniform distribution on $\mathbb{Z}_q^n$ and $(\cdot, \cdot)$ denotes the standard scalar product of $\mathbb{Z}_q^n$. If we are able to determine the scalar product (u, s) for n linearly independent u using the projected states, then s can also be computed using Gaussian elimination. However when q is not large enough compared to n then the error probability for computing (u, s) is too large and we get a system of noisy linear equations for which no efficient algorithms are known. Instead, we can devise a measurement, that at the cost of sacrificing a $1 - 1/q^{O(1)}$ fraction of the samples, yields samples u such that (u, s) belongs to a small subset of $\mathbb{Z}_q$ for sure. More precisely, the samples follow an (A, ℓ) distribution where A is of size $q - r + 1$ and $\ell = (s, \cdot)$. Then we apply Theorem 3 and some easy other steps to determine s.

Paper organization: In Section 2, we provide the algorithm for $\mathrm{LFS}(q, n, d)$ and prove Theorem 3. In Section 3 we propose a Fourier sampling based algorithm for $\mathrm{HMS}(q, n, r)$ and prove Theorem 6. Finally, in Section 4 we reduce $\mathrm{HMS}(q, n, r)$ to $\mathrm{LFS}(q, n, q - r + 1)$ and prove Theorem 5. We provide a complete proof for Theorem 3 here, the proofs for the other results are given in the full version [11].

2 An algorithm for LFS

Let p be an (A,ℓ)-distribution on $\mathbb{F}_q^n$, where $|A| = d$. We define α_p as the element $\alpha \in A$ for which $\Pr[\ell(u) = \alpha]$ is maximal (breaking a tie arbitrarily). We start the proof with our main technical Lemma 8 which links p to the space of n-variable polynomials of degree d.

The proof of Lemma 8 requires the following variant of the Schwartz-Zippel lemma [21, 19] (proved in [11]) where the polynomial $g(x)$ is not divisible by a linear function $\ell(x)$ and the samples are drawn from an affine subspace $V_\alpha = \{u \in \mathbb{Z}_q^n : \ell(u) = \alpha\}$ for a fixed $\alpha \in \mathbb{F}_q$.

▶ **Lemma 7.** *Let $g(x_1 \ldots, x_n)$, be a degree d polynomial in $\mathbb{F}_q[x_1, \ldots, x_n]$ that is not divisible by $\ell(x_1, \ldots, x_n) - \alpha$ where $\alpha \in \mathbb{F}_q$ and $\ell(x_1, \ldots, x_n)$ is a nonzero homogeneous linear polynomial. Let $u = (\beta_1, \ldots, \beta_n)$ be sampled uniformly at random from the affine subspace $V_\alpha = \{u \in \mathbb{Z}_q^n : \ell(u) = \alpha\}$, then $\Pr_{u\sim V_\alpha}[g(u) = 0] \le \frac{d}{q}$.*

▶ **Lemma 8.** *Let $N = \Omega\left(\binom{n+d}{d} d^2 \log \binom{n+d}{d}\right)$ and let $u_1, \ldots, u_N$ be sampled independently from an (A,ℓ)-distribution on $\mathbb{F}_q^n$, where $|A| = d < q$. Then with probability at least $1/2$, every polynomial $g(x_1, \ldots, x_n)$ over $\mathbb{F}_q^n$ of degree at most d, for which $g(u_i) = 0$ for $i = 1, \ldots, N$, is divisible by $\ell(x_1, \ldots, x_n) - \alpha_p$.*

Proof. For $j = 0, \ldots, N$ we set P_j to be the set of polynomials in $\mathbb{F}_q[x_1, \ldots, x_n]$ of degree at most d which take zero value on the first j samples:

$P_j = \{g(x_1, \ldots, x_n) : \deg g \le d \text{ and } g(u_i) = 0 \text{ for } i = 1, \ldots, j\}$.

In particular, P_0 is the set of all polynomials of degree at most d. We consider P_0 as a vector space of dimension $\binom{n+d}{d}$ over $\mathbb{F}_q$. Since, for $u \in \mathbb{F}_q^n$, the map $g \mapsto g(u)$ is linear on $\mathbb{F}_q[x_1, \ldots, x_n]$, we conclude that $P_0, \ldots, P_N$ is a non-increasing sequence of subspaces of P_0.

Set $\pi = \Pr[\ell(u) = \alpha_p]$, and observe that $\pi \ge \frac{1}{d}$. Let P' be the set of polynomials from P_0 which are divisible by $\ell(x_1, \ldots, x_n) - \alpha_p$. Then an equivalent way to state the lemma is that $P_N \subset P'$, with probability at least $1/2$.

We first claim that, for every $j = 1, \ldots, N$,

$$\Pr[P_j = P_{j-1} | P_{j-1} \not\subseteq P'] \le 1 - \frac{1}{d(d+1)}. \tag{1}$$

In order to prove this bound, we note that the condition $P_{j-1} \not\subseteq P'$ means that there exists a non zero $g \in P_{j-1} \setminus P'$. Fix such a g. The event $P_j = P_{j-1}$ is equivalent to $f(u_j) = 0$, for all $f \in P_{j-1}$. Therefore

$$\Pr[P_j = P_{j-1} | P_{j-1} \not\subseteq P'] \le \Pr[\forall f \in P_{j-1},\ f(u_j) = 0] \le \Pr[g(u_j) = 0].$$

The probability that $g(u_j) = 0$ can be bounded as follows:

$$\begin{aligned}\Pr[g(u_j) = 0] &\le \Pr[g(u_j) = 0 | \ell(u_j) \ne \alpha_p] \cdot (1 - \pi) + \Pr[g(u_j) = 0 | \ell(u_j) = \alpha_p] \cdot \pi \\ &\le (1 - \pi) + \pi \frac{d}{q}.\end{aligned}$$

The first inequality follows simply by decomposing the event $g(u_j) = 0$ according to whether $\ell(u_j)$ is different from, or equal to α_p. In the second case, which happens with probability π, Lemma 7 is applicable and it states that $g(u_j) = 0$ with probability at most d/q. This explains the second inequality. Using $\pi \ge 1/d$ and $q \ge d + 1$, a simple calculation gives

$1 - \pi + \pi\frac{d}{q} \le 1 - \frac{1}{d(d+1)}$,

from which the inequality (1) follows.

Algorithm 1 Algorithm for $\mathrm{LFS}(q, n, d)$.

```
Require: A set $A \subset \mathbb{F}_q$ of cardinality $d$ and a sequence of $N$ elements $u_1, \ldots, u_N$ from $\mathbb{F}_q^n$.
  1. Find a nonzero polynomial $g(x_1, \ldots, x_n)$ of degree at most $d$ over $\mathbb{F}_q$, if it exists, such
     that $g(u_i) = 0$ for $i = 1, \ldots, N$.
  2. Compute the linear factors of $g$.
  3. Find a linear factor $f$ of $g$ and a nonzero element $\gamma \in \mathbb{F}_q$, if exist, such that $\gamma(f(u_i) -$
     $f(0)) \in A$, for $i = 1, \ldots, N$. Return the linear function $\gamma(f(x_1, \ldots, x_n) - f(0))$.
```

We can use the conditional probability in (1) to upper bound the probability of the event that $P_{j-1} \not\subseteq P'$ and $P_j = P_{j-1}$ hold simultaneously. But if $P_j = P_{j-1}$ then $P_{j-1} \subseteq P'$ is equivalent to $P_j \subseteq P'$, therefore we can infer, for every $j = 1, \ldots, N$ that $\Pr[P_j \not\subseteq P' \text{ and } P_j = P_{j-1}] \leq 1 - \frac{1}{d(d+1)}$.

Iterating the above argument k-times, we obtain, for every $k \leq N$ and $j \leq N - k + 1$,

$$\Pr[P_{j+k-1} \not\subseteq P' \text{ and } P_{j+k-1} = P_{j-1}] \leq \left(1 - \frac{1}{d(d+1)}\right)^k. \tag{2}$$

Indeed, as before, we can bound the probability on the left hand side by the conditional probability $\Pr[P_{j+k-1} = P_{j-1} | P_{j+k-1} \not\subseteq P']$. Under the condition $P_{j+k-1} \not\subseteq P'$, there exists a non zero $g \in P_{j+k-1} \setminus P'$, and we fix such a g. Then

$$\begin{aligned}\Pr[P_{j+k-1} \not\subseteq P' \text{ and } P_{j+k-1}] &\leq \Pr[g(u_{j+i}) = 0, \text{ for } i = 0, \ldots, k-1] \\ &\leq \prod_{i=0}^{k-1} \Pr[g(u_{j+i}) = 0] \leq (1 - \frac{1}{d(d+1)})^k,\end{aligned}$$

where for the second inequality we used that the samples u_{j+i} are independent.

Taking $k = \Omega(d^2 \log \binom{n+d}{d})$, $N = (\binom{n+d}{d} + 1)k$ and $j = mk + 1$, for $m = 0, 1, \ldots, \binom{n+d}{d}$, in inequality (2), we get $\Pr[P_{(m+1)k} \not\subseteq P' \text{ and } P_{(m+1)k} = P_{mk}] \leq \frac{1}{2}\binom{n+d}{d}^{-1}$.

For the complement of the union of these $\binom{n+d}{d} + 1$ events, we derive then

$$\Pr[\bigcap_{m=0}^{\binom{n+d}{d}} \left(P_{(m+1)k} \subseteq P' \text{ or } P_{(m+1)k} \subset P_{mk}\right)] \geq \frac{1}{2}.$$

If $P_{(m+1)k} \subset P_{mk}$ for some m, then $\dim(P_{(m+1)k}) < \dim(P_{mk})$. We can not have simultaneously $\dim(P_{(m+1)k}) < \dim(P_{mk})$, for $m = 0, 1, \ldots, \binom{n+d}{d}$, because otherwise $\dim(P_N)$ would be negative. Therefore, with probability at least 1/2, $P_{(m+1)k} \subseteq P'$, for some $m \leq \binom{n+d}{d}$, implying $P_N \subseteq P'$. ◀

We now present an algorithm for $\mathrm{LFS}(q, n, d)$ and show that it solves the problem efficiently when the input contains a polynomially large number of samples $u_1, \ldots, u_N \in \mathbb{F}_q^n$ from an (A, ℓ)-distribution, with $|A| = d$ constant.

▶ **Theorem 9.** *There is a randomized implementation of* **Algorithm 1** *which runs in time polynomial in* $\log q$, $\binom{n+d}{d}$ *and* N. *Moreover, when* $u_1, \ldots, u_N$ *are independent samples from an* (A, ℓ)*-distribution on* $\mathbb{F}_q^n$ *where* $|A| = d$ *and* $N = \Omega\left(\binom{n+d}{d} d^2 \log \binom{n+d}{d}\right)$, *then it finds successfully* ℓ *up to a constant factor with probability at least* $1/2$.

Proof. We first describe the randomized implementation with the claimed running time. Throughout the proof by polynomial time we mean time polynomial in $\log q$, $\binom{n+d}{d}$ and N. For Step 1, we consider the $\binom{n+d}{d}$ dimensional vector space of n-variable polynomials over $\mathbb{F}_q$ of degree at most d. The system of requirements $g(u_i) = 0$, for $i = 1, \ldots, N$, is equivalent to a system of N homogeneous linear equations for the $\binom{n+d}{d}$ coefficients of g, where in the ith equation, the coefficients of the variables are the values of the monomials taken at u_i. Therefore a solution, if it exists, can be computed in polynomial time using standard linear algebra.

We use Kaltofen's algorithm [12] (the finite field case is dealt with explicitly in [20]) to find the irreducible factors of g. It is a Las Vegas randomized algorithm, and it runs in polynomial time given the representation of the input polynomial as a list of all coefficients. We can then easily select the linear factors out of the irreducible factors, therefore Step 2 can also be done in polynomial time.

For Step 3, note that g has at most $d \leq n$ linear factors, therefore it is enough to see that each individual factor f can be dealt with in polynomial time. This can be done as follows. If $f(u_i) = f(0)$ for every i, then an appropriate γ can be found if and only if $0 \in A$. Indeed, if $0 \in A$ then any nonzero γ satisfies the condition, while otherwise no satisfying γ exists. Otherwise, pick any i such that $\beta = f(u_i) - f(0) \neq 0$ and try $\gamma = \alpha/\beta$ for every $\alpha \in A$.

We now turn to the proof of correctness of the algorithm when the samples come from an (A, ℓ)-distribution. As $\prod_{\alpha \in A} (\ell(u_i) - \alpha) = 0$, for every i, the algorithm finds a nonzero polynomial g in Step 1. By Lemma 8, with probability at least 1/2, every polynomial of degree at most d, which is zero on u_i, for $i = 1, \ldots, N$, is divisible by $\ell(x_1, \ldots, x_n) - \alpha_p$. Assume that this is the case. Then, in particular, g has a linear factor $f(x)$ which is a constant multiple of $\ell(x) - \alpha_p$, that is $f(x) = \beta(\ell(x) - \alpha_p)$, for some non zero $\beta \in \mathbb{F}_q$. It is easy to check that for $\gamma = \beta^{-1}$, we have $\gamma(f(x) - f(0)) = \ell(x)$, and therefore $\gamma(f(u_i) - f(0)) \in A$, for $i = 1, \ldots, N$. Thus the algorithm in its last step will find successfully and return a linear function $\ell'(x)$ such that $\ell'(u_i) \in A$, for every i.

To finish the proof, we claim that $\ell'(x)$ is a constant multiple of $\ell(x)$. The polynomial $h(x_1, \ldots, x_n) = \prod_{\alpha \in A}(\ell'(x_1, \ldots, x_n) - \alpha)$ is zero on every u_i and hence, by our assumption, $h(x_1, \ldots, x_n)$ is divisible by $\ell(x_1, \ldots, x_n) - \alpha_p$. Then, as $\mathbb{F}_q[x_1, \ldots, x_n]$ is a unique factorization domain, there exists $\alpha \in A$ such that $\ell'(x_1, \ldots, x_n) - \alpha$ is a scalar multiple of $\ell(x_1, \ldots, x_n) - \alpha_p$, implying the claim. ◀

Theorem 3 is an immediate consequence of this result. For constant d we have the following corollary.

▶ **Corollary 10.** *There is a randomized algorithm that solves* LFS(q, n, d) *for constant d with sample complexity* poly(n) *and running time* poly$(n, \log q)$.

We next present our algorithms for HMS(q, n, r), we first give a basic Fourier sampling based algorithm in section 3 and then an algorithm that reduces HMS(q, n, r) to LFS$(q, n, q-r+1)$ in section 4.

3 Fourier sampling algorithm for HMS(q, n, r)

We first describe briefly the standard pre-processing procedure for HMS(q, n, r). starting with the uniform superposition, append a register consisting of l qubits, initialized to 0 and query the oracle for f_s to obtain,

$$\frac{1}{\sqrt{q^n r}} \sum_{v \in \mathbb{Z}_q^n} \sum_{h \in H} |v\rangle |h\rangle \rightarrow \frac{1}{\sqrt{q^n r}} \sum_{v \in \mathbb{Z}_q^n} \sum_{h \in H} |v\rangle |h\rangle |f_s(v, h)\rangle .$$

The last l qubits are then measured to obtain the state,

$$\psi_s^w := \frac{1}{\sqrt{r}} \sum_{h \in H} |w + hs\rangle |h\rangle ,$$

where $w \in \mathbb{Z}_q^n$ is uniformly random. This w is the unique element of $\mathbb{Z}_q^n$ such that the measured value for the function f_s equals $f(w)$.

It is standard to then apply the quantum Fourier transform on $\mathbb{Z}_q^n$ to the states ψ_s^w and to measure the first register to obtain tuples (u, ϕ_s^u) where $u \in \mathbb{Z}_q^n$ is uniformly random and $\phi_s^u := \frac{1}{\sqrt{r}} \sum_{h \in H} \omega^{(u,hs)} |h\rangle$. We therefore assume without loss of generality that the quantum input for $\mathrm{HMS}(q,n,r)$ are N samples of the form (u, ϕ_s^u) for uniformly random $u \in \mathbb{Z}_q^n$.

We next give the Fourier sampling based algorithm for $\mathrm{HMS}(q,n,r)$ and prove Theorem 6. The basic idea for the algorithm is to consider the input state $\phi_s^u = \frac{1}{\sqrt{r}} \sum_{h \in H} \omega^{(u,hs)} |h\rangle$ for $\mathrm{HMS}(q,n,r)$, as an approximation to the state $\kappa_s^u := \frac{1}{\sqrt{q}} \sum_{h=0}^{q-1} \omega^{(u,hs)} |h\rangle$. The inner product between the two states is $\phi_s^{u\dagger} \cdot \kappa_s^u = \frac{1}{\sqrt{qr}} \sum_{h \in H} 1 = \sqrt{r/q}$.

The inverse Fourier transform on $\mathbb{Z}_q$, when applied to κ_s^u gives $|(u,s)\rangle$. If we could determine the inner products $|(u,s)\rangle$ for a set of n linearly independent u_i for prime q, then s can be determined by solving a system of linear equations. More generally, in order to make this approach work k should be large enough so that the u_i generate $\mathbb{Z}_q^n$, in this case the secret s can be recovered from the inner products using linear algebra. In fact, the following result from [16] shows that the additive group $\mathbb{Z}_q^n$ is generated by $k = n + O(1)$ random elements of $\mathbb{Z}_q^n$ with constant probability.

▶ **Fact 11.** *[16] Let G be a finite abelian group with a minimal generating set of size r. The expected number of elements chosen independently and uniformly at random from G such that the chosen elements generate G is at most $r + \sigma$ where $\sigma < 2.12$ is an explicit constant.*

The above fact holds for any abelian group, for the special case of $\mathbb{Z}_q^n$ we have $r = n$ and the constant σ can be taken to be 1 [1]. We therefore have that $k = 2n + O(1)$ random elements of $\mathbb{Z}_q^n$ generate the additive group $\mathbb{Z}_q^n$ with constant probability.

If we apply the Fourier transform to each $\phi_s^{u_i}$, with probability $(r/q)^{k/2}$ we obtain the scalar products of s with the members of a generating set for $\mathbb{Z}_q^n$. The answer s may be verified by repeating the experiment for $\mathrm{poly}(n)(q/r)^{k/2}$ trials and finding the most frequently occurring solutions over the different trials.

▶ **Theorem 6.** *There is a quantum algorithm that solves* $\mathrm{HMS}(q,n,r)$ *with high probability in time* $O(\mathrm{poly}(n)(\frac{q}{r})^{n+O(1)})$.

We next show that the above algorithm runs in time $\mathrm{poly}(n)$ for parameters q, r such that $\frac{r}{q} = 1 - \Omega(\frac{\log n}{n})$. For this choice of parameters, we can bound the factor $(\frac{q}{r})^{n+O(1)}$ in the running time bound above as follows,

$$\left(\frac{r}{q}\right)^{n+O(1)} \geq \left(1 - \frac{c_1 \log n}{n}\right)^{c_2 n + c_3} \geq e^{-c \log n} = n^{-O(1)}$$

where c, c_1, c_2 are suitable constants. We therefore have,

▶ **Corollary 12.** *If $\frac{r}{q} = 1 - \Omega(\frac{\log n}{n})$, then there is a quantum algorithm that solves* $\mathrm{HMS}(q,n,r)$ *with high probability in time* $\mathrm{poly}(n)$.

4 Reducing $\mathrm{HMS}(q,n,r)$ to $\mathrm{LFS}(q,n,q-r+1)$

In this section we assume that q is a prime number and work over the field $\mathbb{F}_q$. Recall that the input for $\mathrm{HMS}(q,n,r)$ is a collection of samples of vector-state pairs (u, ϕ_s^u) where u is a uniformly random vector from $\mathbb{F}_q^n$, and $\phi_s^u = \frac{1}{\sqrt{r}} \sum_{h \in H} \omega^{(u,s)h} |h\rangle$. For $t \in \mathbb{F}_q$ define the state $\mu_t := \frac{1}{\sqrt{r}} \sum_{h \in H} \omega^{ht} |h\rangle$, so that $\phi_s^u = \mu_{(u,s)}$.

The approach in Section 3 recovers the inner product (u_i, s) for $O(n)$ random vectors u_i and then uses Gaussian elimination to determine s with high probability. However, the μ_t's are only nearly orthogonal to each other, so the measurement in Section 3 may fail to recover the correct value of (u, s) with probability too large for our purposes.

A particularly interesting case is when $q = \text{poly}(n)$ and $c = q - r$ is a constant for which we provide a polynomial time algorithm in Corollary 20. In this case, the error probability for the measurement in Section 3 is $1 - r/q = c/q$, that is there are a constant expected number of errors for every q samples. If $q = O(n^{\alpha})$ for $\alpha < 1$ then there are $O(n^{1-\alpha})$ errors in expectation for every n samples. There are no known polynomial time algorithms for recovering the secret $s \in \mathbb{Z}_q^n$ from a system of n linear equations where an $O(n^{1-\alpha})$ fraction of the equations are incorrect for a constant α.

Instead, we reduce $\text{HMS}(q, n, r)$ to $\text{LFS}(q, n, d)$ with $d = (q-r)+1$, $A = \{r-1, \ldots, q-1\}$ and the linear function $\ell(\cdot)$ given by $\ell(x) = (s, x)$. We then use the Algorithm 1 to recover a scalar multiple of $s_0 = \lambda s$. Further, we show that the scalar λ can be recovered efficiently.

The reduction performs a quantum measurement on ϕ_s^u to determine if (u, s) belongs to $A = \{r-1, \ldots, q-1\}$. We discard the u's which do not belong to A, and also some of the u's such that $(u, s) \in A$ to obtain samples from an (A, ℓ) distribution. We next provide a sketch of the reduction from $\text{HMS}(q, n, r)$ to $\text{LFS}(q, n, d)$, the reduction is analyzed over the next few subsections and a more precise statement is given in Proposition 17.

Let V be the hyperplane spanned by $\mu_0, \ldots, \mu_{r-2}$. Let (u, ϕ_s^u) be a pair from the input samples. We perform the measurement on ϕ_s^u according to the decomposition of $\mathbb{C}^r = V \oplus V^{\perp}$, and retain u if and only if the result of the measurement is 'in $V^{\perp}$. Otherwise we discard u. An efficient implementation of the measurement in $(V, V^{\perp})$ is given later in this section.

Observe that measuring a state μ_j 'in $V^{\perp}$ is only possible if $\mu_j \notin V$, in particular $j \notin \{0, \ldots, r-2\}$. Thus if we measure $\phi_s(u)$ 'in $V^{\perp}$ we can be sure that (s, u) is in $A = \{r-1, \ldots, q-1\}$. We only keep u from a sample pair (u, τ) if this measurement, applied to the state τ, results 'in $V^{\perp}$'. The u's that are retained are samples from an (A, ℓ) distribution over $\mathbb{F}_q^n$.

We bound the probability of retaining a sample pair (u, ϕ_s^u) for this procedure. We bound the success probability for the special case when $(s, u) = r - 1$. As u is uniformly random over $\mathbb{F}_q^n$ the value of (s, u) is uniformly distributed over $\mathbb{Z}_q$, this bound therefore suffices for our purposes.

In the standard basis $|h\rangle$ $(h \in H)$, the vector μ_t has entry $\frac{1}{\sqrt{r}}\omega^{ht}$ in the h-th position. Let $A \in \mathbb{C}^{r \times r}$ be the matrix with rows from the collection $\{\mu_t : 0 \le t \le r-1\}$, that is

$$A = \frac{1}{\sqrt{r}} \begin{pmatrix} 1 & \omega^{h_1} & \cdots & \omega^{h_1(r-1)} \\ 1 & \omega^{h_2} & \cdots & \omega^{h_2(r-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{h_r} & \cdots & \omega^{h_r(r-1)} \end{pmatrix}, \tag{3}$$

where $h_1, \ldots, h_r$ are the elements of H, say, in increasing order. The matrix A is $\frac{1}{\sqrt{r}}$ times a Vandermonde matrix and as such, it is well known that it has determinant $r^{-r/2} \prod_{j<i\le r}(\omega^{h_i} - \omega^{h_j})$. In particular, the states $\mu_0, \ldots, \mu_{r-1}$ are linearly independent. With a more careful analysis, we show in Lemma 13 below that $\det A$ is sufficiently far from zero.

▶ **Lemma 13.** *Let $c = q - r$ and A be the matrix in* (3) *then* $|\det A^* A| = \Omega(q^{-c^2} \left(\frac{q}{r}\right)^r)$.

Using the above lemma, we bound the probability of retaining u if $(u, s) = r - 1$. It might be possible to prove similar bounds for other values of $(s, u) \in A$, however the bound for the particular value $r - 1$ suffices for our purpose.

▶ **Lemma 14.** *The $(V, V^{\perp})$-measurement applied to a state of the form μ_t, returns "in V" with probability 1 if $t \in \{0, \ldots, r-2\}$, while for $t \in \{r-1, \ldots, q-1\}$, the probability that "in $V^{\perp}$" is returned depends only on t and is $\Omega\left(q^{-c^2}\left(\frac{q}{r}\right)^r\right)$ for $t = r-1$.*

We describe next the implementation of the $(V, V^{\perp})$ measurement that acts on $O(\log q)$ qubits. Using universality constructions and the Solovay-Kitaev theorem, it is well known that an arbitrary unitary operator can be approximated using an exponential number of elementary gates in the number of qubits.

▶ **Fact 15.** *[14] An arbitrary unitary operation U on t qubits can be simulated to error ϵ using $O(t^2 4^t \log^c(t^2 4^t/\epsilon))$ elementary gates.*

The ability to implement an arbitrary unitary operation on $\log q$ qubits implies the ability to perform the measurement $(W, W^{\perp})$ for an arbitrary subspace $W \subset \mathbb{C}^q$.

Denote the quantum state corresponding to unit vector $w \in \mathbb{C}^q$ as $|w\rangle := \sum_{i=1}^{q} w_i |i\rangle$. Let k be the dimension of W and let $w_1, w_2, \cdots, w_k$ be an orthonormal basis for W. Let U_W be a unitary operation that maps the standard basis vectors $|i\rangle \to |w_i\rangle$. Then the measurement in $(W, W^{\perp})$ on state $|\phi\rangle$ can be implemented by first computing $U_W^{-1}|\phi\rangle$ and then measuring in the standard basis. The state $|\phi\rangle$ belongs to W if and only if the result of measurement in the standard basis belongs to the set $\{1, 2, \cdots, k\}$. As the $(V, V^{\perp})$ measurement is on $\log q$ qubits, by Fact 15 we have,

▶ **Claim 16.** *The measurement $(V, V^{\perp})$ can be implemented to precision $1/q^{O(1)}$ in time $\widetilde{O}(q^2)$.*

The implementation of the $(V, V^{\perp})$ measurement above shows that the sampling procedure can be performed efficiently. The procedure yields a sample from an (A, ℓ) distribution with $|A| = (q-r)+1$ when the measurement outcome is $V^{\perp}$. By Lemma 14 the outcome $V^{\perp}$ occurs with probability $\Omega\left(q^{-c^2}\left(\frac{q}{r}\right)^r\right)$ if $(u, s) = r-1$. As u is uniformly random on $\mathbb{F}_q^n$ at least a $\Omega\left(q^{-c^2-1}\left(\frac{q}{r}\right)^r\right)$ fraction of the samples are retained. We therefore have the following proposition,

▶ **Proposition 17.** *There is a quantum procedure that that runs in time $\widetilde{O}(q^2)$, and given a pair (u, ϕ_s^u) where $u \in \mathbb{Z}_q^n$ is uniformly random and $\phi_s^u = \mu_{(u,s)}$, with probability at least $O\left(q^{-c^2-1}\left(\frac{q}{r}\right)^r\right)$ returns a sample from a (A, ℓ) distribution with $|A| = c+1$ and $\ell(x) = (s, x)$.*

In order to solve $\text{HMS}(q, n, r)$ given a scalar multiple $s_0 = \lambda s$ found using Theorem 9, we need to find the scalar λ. We show that using $O(q)$ further input pairs we can find the value of λ using a simple trial and error procedure given by the following lemma.

▶ **Lemma 18.** *Given $t \in \mathbb{F}_q$ and a state $\tau \in \mathbb{C}^r$, there is a quantum procedure that returns YES with probability 1 if $\tau = \mu_t$, while if $\tau = \mu_{t'}$ for some $t' \in \mathbb{F}_q \setminus \{t\}$, it returns YES with probability at most* $p := \begin{cases} 1/4 & \textit{if } q < 3r/2 \\ 1 - O(1/q) & \textit{otherwise.} \end{cases}$

Combining the results proved in this section with Algorithm 1, we next obtain a quantum algorithm for $\text{HMS}(q, n, r)$ for the case of prime q.

Theorem 9 shows that given $N = \Omega(\binom{n+d}{d} d^2 \log(\binom{n+d}{d}))$ samples from an (A, ℓ) distribution, a scalar multiple of the function ℓ can be found with constant probability. Proposition 17 above shows that the expected time to obtain N samples from the (A, ℓ) distribution is

$\widetilde{O}(Nq^{c^2+3})$ where we used that each measurment requires time $\widetilde{O}(q^2)$ and ignored the factor $(r/q)^r < 1$. The number of samples and the time required for determining the scalar λ in Lemma18 are negligible compared to these quantities. We therefore have the following theorem,

▶ **Theorem 19.** *Let q be a prime and let $c = q - r$. Then there is a quantum algorithm which solves* $\mathrm{HMS}(q, n, r)$ *with sample complexity* $\widetilde{O}(n^{c+1}q^{c^2+1})$ *and in time* $\widetilde{O}(n^{c+1}q^{c^2+3})$.

The algorithm runs in time polynomial in $n, \log q$ for the case when $q = \mathrm{poly}(n)$. We therefore we have the following corollary,

▶ **Corollary 20.** *Let $q = \mathrm{poly}(n)$ be a prime number and $c = q - r$ be a constant, then there is an efficient quantum algorithm for* $\mathrm{HMS}(q, n, r)$.

References

1 Vincenzo Acciaro. The probability of generating some common families of finite groups. *Utilitas Math.*, 49:243–254, 1996.

2 Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming ICALP, Zurich, Switzerland*, pages 403–415. Springer, 2011.

3 Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM (JACM)*, 50(4):506–519, 2003.

4 Andrew M. Childs, Robin Kothari, Maris Ozols, and Martin Roetteler. Easy and hard functions for the boolean hidden shift problem. In Simone Severini and Fernando G. S. L. Brandão, editors, *8th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2013, May 21-23, 2013, Guelph, Canada*, volume 22 of *LIPIcs*, pages 50–79. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. `doi:10.4230/LIPIcs.TQC.2013.50`.

5 Andrew M. Childs and Wim van Dam. Quantum algorithm for a generalized hidden shift problem. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 1225–1232. SIAM, 2007. URL: `http://dl.acm.org/citation.cfm?id=1283383.1283515`.

6 Thomas Decker, Gábor Ivanyos, Raghav Kulkarni, Youming Qiao, and Miklos Santha. An efficient quantum algorithm for finding hidden parabolic subgroups in the general linear group. In *International Symposium on Mathematical Foundations of Computer Science*, pages 226–238. Springer, 2014.

7 Mark Ettinger and Peter Høyer. On quantum algorithms for noncommutative hidden subgroups. *Advances in Applied Mathematics*, 25:239–251, 2000. `arXiv:arXiv:quant-ph/9807029`.

8 Katalin Friedl, Gábor Ivanyos, Frédéric Magniez, Miklos Santha, and Pranab Sen. Hidden translation and translating coset in quantum computing. *SIAM Journal on Computing*, 43(1):1–24, 2014. preliminary version in STOC 2003.

9 Dmitry Gavinsky, Martin Roetteler, and Jérémie Roland. Quantum algorithm for the boolean hidden shift problem. In Bin Fu and Ding-Zhu Du, editors, *Computing and Combinatorics - 17th Annual International Conference, COCOON 2011, Dallas, TX, USA, August 14-16, 2011. Proceedings*, volume 6842 of *Lecture Notes in Computer Science*, pages 158–167. Springer, 2011. `doi:10.1007/978-3-642-22685-4_14`.

10 Gábor Ivanyos. On solving systems of random linear disequations. *Quantum Information & Computation*, 8(6):579–594, 2008. URL: `http://www.rintonpress.com/xxqic8/qic-8-67/0579-0594.pdf`.

11 Gábor Ivanyos, Anupam Prakash, and Miklos Santha. On learning linear functions from subset and its applications in quantum computing. *arXiv*, 2018. arXiv:1710.02581.

12 Erich Kaltofen. Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization. *SIAM J. Comput.*, 14(2):469–489, 1985. doi:10.1137/0214035.

13 Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM Journal on Computing*, 35(1):170–188, 2005. arXiv:quant-ph/0302112.

14 Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

15 Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 333–342. ACM, 2009.

16 Carl Pomerance. The expected number of random elements to generate a finite abelian group. *Periodica Mathematica Hungarica*, 43(1):191–198, Aug 2002. doi:10.1023/A:1015250102792.

17 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

18 Martin Roetteler. Quantum algorithms for abelian difference sets and applications to dihedral hidden subgroups. In Anne Broadbent, editor, *11th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC, September 27-29, 2016, Berlin, Germany*, volume 61 of *LIPIcs*, pages 8:1–8:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.TQC.2016.8.

19 Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980.

20 Joachim von zur Gathen and Erich Kaltofen. Polynomial-time factorization of multivariate polynomials over finite fields. In *International Colloquium on Automata, Languages, and Programming*, pages 250–263. Springer, 1983.

21 Richard Zippel. Probabilistic algorithms for sparse polynomials. *Symbolic and algebraic computation*, pages 216–226, 1979.

Strong Collapse for Persistence

Jean-Daniel Boissonnat
Université Côte d'Azur, INRIA, France
Jean-Daniel.Boissonnat@inria.fr

Siddharth Pritam
Université Côte d'Azur, INRIA, France
siddharth.pritam@inria.fr

Divyansh Pareek
Indian Institute of Technology Bombay, India
divyansh@cse.iitb.ac.in

Abstract

We introduce a fast and memory efficient approach to compute the persistent homology (PH) of a sequence of simplicial complexes. The basic idea is to simplify the complexes of the input sequence by using strong collapses, as introduced by J. Barmak and E. Miniam [DCG (2012)], and to compute the PH of an induced sequence of reduced simplicial complexes that has the same PH as the initial one. Our approach has several salient features that distinguishes it from previous work. It is not limited to filtrations (i.e. sequences of nested simplicial subcomplexes) but works for other types of sequences like towers and zigzags. To strong collapse a simplicial complex, we only need to store the maximal simplices of the complex, not the full set of all its simplices, which saves a lot of space and time. Moreover, the complexes in the sequence can be strong collapsed independently and in parallel. As a result and as demonstrated by numerous experiments on publicly available data sets, our approach is extremely fast and memory efficient in practice.

2012 ACM Subject Classification Mathematics of computing → Algebraic topology

Keywords and phrases Computational Topology, Topological Data Analysis, Strong Collapse, Persistent homology

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.67

Funding This research has received funding from the European Research Council (ERC) under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement No. 339025 GUDHI (Algorithmic Foundations of Geometry Understanding in Higher Dimensions).

Acknowledgements We want to thank Marc Glisse for his useful discussions, Mathijs Wintraecken for reviewing a draft of the article. We also want to thank Francois Godi and Siargey Kachanovich for their help with Gudhi and Hannah Schreiber for her help with Sophia.

1 Introduction

In this article, we address the problem of computing the Persistent Homology (PH) of a given sequence of simplicial complexes (defined precisely in Section 4) in an efficient way. It is known that computing persistence can be done in $O(n^\omega)$ time, where n is the total number of simplices and $\omega \leq 2.4$ is the matrix multiplication exponent [20, 15]. In practice, when dealing with massive and high-dimensional datasets, n can be very large (of order of billions) and computing PH is then very slow and memory intensive. Improving the performance

© Jean-Daniel Boissonnat, Siddharth Pritam, and Divyansh Pareek;
licensed under Creative Commons License CC-BY

of PH computation has therefore become an important research topic in Computational Topology and Topological Data Analysis.

Much progress has been accomplished in the recent years in two directions. First, a number of clever implementations and optimizations have led to a new generation of software for PH computation [16, 27, 24, 13]. Secondly, a complementary direction has been explored to reduce the size of the complexes in the sequence while preserving (or approximating in a controlled way) the persistent homology of the sequence. Examples are the work of Mischaikow and Nanda [21] who use Morse theory to reduce the size of a filtration, and the work of Dłotko and Wagner who use simple collapses [14]. Both methods compute the exact PH of the input sequence. Approximations can also be computed with theoretical guarantees. Approaches like interleaving smaller and easily computable simplicial complexes, and sub-sampling of the point sample works well upto certain approximation factor [8, 5, 25, 19, 9, 12].

In this paper, we introduce a new approach to simplify the complexes of the input sequence which uses the notion of strong collapse introduced by J. Barmak and E. Miniam [2]. Specifically, our approach can be summarized as follows. Given a sequence $\mathcal{Z} : \{K_1 \xrightarrow{f_1} K_2 \xleftarrow{g_2} K_3 \xrightarrow{f_3} \cdots \xrightarrow{f_{(n-1)}} K_n\}$ of simplicial complexes K_i connected through simplicial maps $\{\xrightarrow{f_i} \text{ or } \xleftarrow{g_j}\}$, we independently strong collapse the complexes of the sequence to reach a sequence $\mathcal{Z}^c : \{K_1^c \xrightarrow{f_1^c} K_2^c \xleftarrow{g_2^c} K_3^c \xrightarrow{f_3^c} \cdots \xrightarrow{f_{(n-1)}^c} K_n^c\}$, with *induced simplicial maps* $\{\xrightarrow{f_i^c}$ or $\xleftarrow{g_j^c}\}$ (defined in Section 4). The complex K_i^c is called the **core** of the complex K_i and we call the sequence $\mathcal{Z}^c$ the **core sequence** of $\mathcal{Z}$. We show that one can compute the PH of the sequence $\mathcal{Z}$ by computing the PH of the core sequence $\mathcal{Z}^c$, which is of much smaller size.

Our method has some similarity with the work of Wilkerson et. al. [29] who also use strong collapses to reduce PH computation but it differs in three essential aspects: it is not limited to filtrations (i.e. sequences of nested simplicial subcomplexes) but works for other types of sequences like towers and zigzags. It also differs in the way strong collapses are computed and in the manner PH is computed.

A first central observation is that to strong collapse a simplicial complex K, we only need to store its maximal simplices (i.e. those simplices that have no coface). The number of maximal simplices is smaller than the total number of simplices by a factor that is exponential in the dimension of the complex. It is linear in the number of vertices for a variety of complexes [4]. Working only with maximal simplices dramatically reduces the time and space complexities compared to the algorithm of [30]. We prove that the complexity of our algorithm is $\mathcal{O}(v^2\Gamma_0 d + m^2\Gamma_0 d)$. Here d is the dimension of the complex, v is the number of vertices, m is the number of maximal simplices and Γ_0 is an upper bound on the number of maximal simplices incident to a vertex. As observed in [3, 4], usually m is much smaller than the total number of simplices and Γ_0 is much smaller than m (see Section 3 for a discussion).

We now consider PH computation. All PH algorithms take as input a full representation of the complexes. We thus have to convert the representation by maximal simplices used for strong collapses into a full representation of the complexes, which takes exponential time in the dimension (of the collapsed complexes). This exponential burden is to be expected since it is known that computing PH is NP-hard when the complexes are represented by their maximal faces [1]. Nevertheless, we demonstrate in this paper that strong collapses combined with known persistence algorithms lead to major improvements over previous methods to compute the PH of a sequence. This is due in part to the fact that strong collapses reduce the size of the complexes on which persistence is computed. Two other factors also play a role:

- The collapses of the complexes in the sequence can be performed independently and in parallel. This is due to the fact that strong collapses can be expressed as simplicial maps unlike simple collapses [28].
- The size of the complexes in a sequence does not grow by much in terms of maximal simplices, as observed in many practical cases. As a consequence, the time to collapse the i-th simplicial complex K_i in the sequence is almost independent of i. For filtrations, this is a clear advantage over methods that use a full representation of the complexes and suffer an increasing cost as i increases.

As a result, our approach is extremely fast and memory efficient in practice as demonstrated by numerous experiments on publicly available data sets.

An outline of this paper is as follows. Section 2 recalls the basic ideas and constructions related to simplicial complexes and strong collapses. We describe our core algorithm in Section 3. In Section 4, we prove that zigzag modules are preserved under strong collapse. In Section 5, we provide experimental results.

2 Preliminaries

In this section, we provide a brief review of the notions of simplicial complex and strong collapse as introduced in [2]. We assume some familiarity with basic concepts like homotopic maps, homotopy type, homology groups and other algebraic topological notions. Readers can refer to [17] for a comprehensive introduction of these topics.

Simplex, simplicial complex and simplicial map. An **abstract simplicial complex** K is a collection of subsets of a non-empty finite set X, such that for every subset A in K, all the subsets of A are in K. From now on we will call an *abstract simplicial complex* simply a *simplicial complex* or just a *complex*. An element of K is called a **simplex**. An element of cardinality $k+1$ is called a k-simplex and k is called its **dimension**. A simplex is called **maximal** if it is not a proper subset of any other simplex in K. A sub-collection L of K is called a **subcomplex**, if it is a simplicial complex itself. L is a **full subcomplex** if it contains all the simplices of K that are spanned by the vertices (0-simplices) of the subcomplex L.

A vertex to vertex map $\psi : K \to L$ between two simplicial complexes is called a **simplicial map**, if the images of the vertices of a simplex always span a simplex. Simplicial maps are thus determined by the images of the vertices. In particular, there is a finite number of simplicial maps between two given finite simplicial complexes. Simplicial maps induce continuous maps between the underlying *geometric realisations* of the simplicial complexes. Two simplicial maps $\phi : K \to L$ and $\psi : K \to L$ are contiguous if, for all $\sigma \in K$, $\phi(\sigma) \cup \psi(\sigma) \in L$. Two contiguous maps are known to be homotopic [22, Theorem 12.5].

Dominated vertex. Let σ be a simplex of a simplicial complex K, the **closed star** of σ in K, $st_K(\sigma)$ is a subcomplex of K which is defined as follows, $st_K(\sigma) := \{\tau \in K | \ \tau \cup \sigma \in K\}$. The **link** of σ in K, $lk_K(\sigma)$ is defined as the set of simplices in $st_K(\sigma)$ which do not intersect with σ, $lk_K(\sigma) := \{\tau \in st_K(\sigma) | \tau \cap \sigma = \emptyset\}$.

Taking a join with a vertex transforms a simplicial complex into a simplicial cone. Formally if L is a simplicial complex and a is a vertex not in L then the **simplicial cone** aL is defined as $aL := \{a, \tau \,|\ \tau \in L \ \ or \ \ \tau = \sigma \cup a; \ \ where \ \ \sigma \in L\}$. A vertex v in K is called a **dominated vertex** if the link of v in K, $lk_K(v)$ is a simplicial cone, that is, there exists a vertex $v' \neq v$ and a subcomplex L in K, such that $lk_K(v) = v'L$. We say that the vertex v'

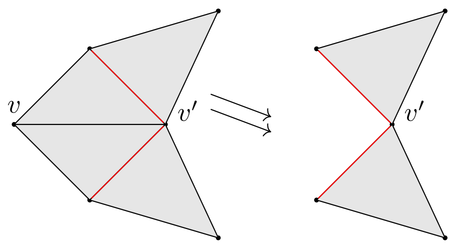


Figure 1 Illustration of an *elementary strong collapse*. In the complex on the left, v is dominated by v'. The link of v is highlighted in red. Removing v leads to the complex on the right.

is *dominating* v and v is *dominated* by v'. The symbol $\boldsymbol{K \setminus v}$(deletion of v from K) refers to the subcomplex of K which has all simplices of K except the ones containing v. Below is an important remark from [2, Remark 2.2], which proposes an alternative definition of dominated vertices.

▶ Remark (1). A vertex $v \in K$ is dominated by another vertex $v' \in K$, *if and only if* all the maximal simplices of K that contain v also contain v' [2].

Strong collapse. An **elementary strong collapse** is the deletion of a dominated vertex v from K, which we denote with $K \searrow\searrow^{e} K \setminus v$. Figure 1 illustrates an easy case of an elementary strong collapse. There is a **strong collapse** from a simplicial complex K to its subcomplex L, if there exists a series of elementary strong collapses from K to L, denoted as $K \searrow\searrow L$. The inverse of a strong collapse is called a **strong expansion**. If there exists a combination of strong collapses and/or strong expansion from K to L then K and L are said to have the same **strong homotopy type**.

The notion of strong homotopy type is stronger than the notion of simple homotopy type in the sense that if K and L have the same strong homotopy type, then they have the same simple homotopy type, and therefore the same homotopy type [2]. There are examples of contractible or simply collapsible simplicial complexes that are not strong collapsible.

A complex without any dominated vertex will be called a **minimal complex**. A **core** of a complex K is a minimal subcomplex $K^c \subseteq K$, such that $K \searrow\searrow K^c$. *Every simplicial complex has a* ***unique core*** *up to isomorphism. The core decides the strong homotopy type of the complex*, and two simplicial complexes have the same strong homotopy type *if and only if* they have isomorphic cores [2, Theorem 2.11].

Retraction map. If a vertex $v \in K$ is dominated by another vertex $v' \in K$, the vertex map $r : K \to K \setminus v$ defined as: $r(w) = w$ if $w \neq v$ and $r(v) = v'$, induces a simplical map that is a *retraction* map. The homotopy between r and the identity $i_{K\setminus v}$ over $K \setminus v$ is in fact a strong deformation retract. Furthermore, the composition $(i_{K\setminus v})r$ is contiguous to the identity i_K over K [2, Proposition 2.9].

Nerve of a simplicial complex. A closed **cover** $\mathcal{U}$ of a topological space $\mathcal{X}$ is a set of closed sets of $\mathcal{X}$ such that $\mathcal{X}$ is a subset of their union. The **nerve** of a cover $\mathcal{U}$ is an abstract simplicial complex, defined as the set of all non-empty intersections of the elements of $\mathcal{U}$. The nerve is a well known construction that transforms a continuous space to a combinatorial space preserving its homotopy type. The *nerve* $\mathcal{N}(K)$ of a simplicial complex K is defined as the nerve of the set of maximal simplices of the complex K (considered as a cover of the complex). Hence all the maximal simplices of K will be the vertices of $\mathcal{N}(K)$ and their

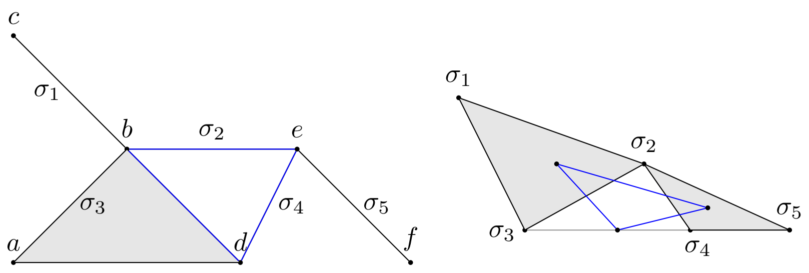


Figure 2 Left: K (in grey), Right: $\mathcal{N}(K)$ (in grey) and $\mathcal{N}^2(K)$ (in blue). $\mathcal{N}^2(K)$ is isomorphic to a full-subcomplex of K highlighted in blue on the left.

non-empty intersection will form the simplices of $\mathcal{N}(K)$. For $j \geq 2$ the iterative construction is defined as $\mathcal{N}^j(K) = \mathcal{N}(\mathcal{N}^{j-1}(K))$. This definition of nerve preserves the homotopy type, $K \simeq \mathcal{N}(K)$[2]. A remarkable property of this nerve construction is its connection with strong collapses.

Taking the nerve of any simplicial complex K twice corresponds to a strong collapse.

▶ **Theorem 1.** *[2, Proposition 3.4] For a simplicial complex K, there exists a subcomplex L isomorphic to $\mathcal{N}^2(K)$, such that $K \searrow\searrow L$.*

An easy consequence of this theorem is that a complex K is *minimal* if and only if it is isomorphic to $\mathcal{N}^2(K)$ [2, Lemma 3.6]. This means that we can keep collapsing our complex K by applying $\mathcal{N}^2(.)$ iteratively until we reach the core of the complex K. The sequence $K, \mathcal{N}^2(K), ..., \mathcal{N}^{2p}(K)$ is a decreasing sequence in terms of number of simplices.

3 Algorithm

In this section, we describe an algorithm to strong collapse a simplicial complex K, provide the details of the implementation and analyze its complexity. We construct $\mathcal{N}^2(K)$ as defined in Section 2.

Data structure. Basically, we represent K as the adjacency matrix M between the vertices and the maximal simplices of K. We will simply call M the adjacency matrix of K. The rows of M represent the vertices and the columns represent the maximal simplices of K. For convenience, we will identify a row (resp. column) and the vertex (resp. maximal simplex) it represents. An entry $M[v_i][\sigma_j]$ associated with a vertex v_i and a maximal simplex σ_j is set to 1 if $v_i \in \sigma_j$, and to 0 otherwise. For example, the matrix M in the left of the Table 1 corresponds to the leftmost simplicial complex K in Figure 2. Usually, M is very sparse. Indeed, each column contains at most $d+1$ non-zero elements since the simplices of a d-dimensional complex have at most $d+1$ vertices, and each line contains at most Γ_0 non-zero elements where Γ_0 is an upper bound on the number of maximal simplices incident to a given vertex. As already mentionned, in many practical situations, Γ_0 is a small fraction of the number of maximal simplices. It is therefore beneficial to store M as a list of vertices and a list of maximal simplices. Each vertex v in the list of vertices points to the maximal simplices that contain v, and each simplex in the list of maximal simplices points to its vertices. This data structure is similar to the SAL data structure of [3].

Table 1 From left to right M, $\mathcal{N}(M)$ and $\mathcal{N}^2(M)$.

	σ_1	σ_2	σ_3	σ_4	σ_5
a	0	0	1	0	0
b	1	1	1	0	0
c	1	0	0	0	0
d	0	0	1	1	0
e	0	1	0	1	1
f	0	0	0	0	1

	b	d	e
σ_1	1	0	0
σ_2	1	0	1
σ_3	1	1	0
σ_4	0	1	1
σ_5	0	0	1

	σ_2	σ_3	σ_4
b	1	1	0
d	0	1	1
e	1	0	1

Core algorithm. Given the adjacency matrix M of K, we compute the adjacency matrix C of the *core* K^c. It turns out that using basic row and column removal operations, we can easily compute C from M. Loosely speaking our algorithm recursively computes $\mathcal{N}^2(K)$ until it reaches K^c.

The columns of M (which represent the maximal simplices of K) correspond to the vertices of $\mathcal{N}(K)$. Also, the columns of M that have a non-zero value in a particular row v correspond to the maximal simplices of K that share the vertex associated with row v. Therefore, each row of M represents a simplex of the nerve $\mathcal{N}(K)$. Not all simplices of $\mathcal{N}(K)$ are associated with rows of M but all maximal simplices are since they correspond to subsets of maximal simplices with a common vertex. To remedy this situation, we *remove* all the rows of M that correspond to non-maximal simplices of $\mathcal{N}(K)$. This results in a new smaller matrix M whose transpose, noted $\mathcal{N}(M)$, is the adjacency matrix of the nerve $\mathcal{N}(K)$. We then exchange the roles of rows and columns (which is the same as taking the transpose) and run the very same procedure as before so as to obtain the adjacency matrix $\mathcal{N}^2(M)$ of $\mathcal{N}^2(K)$.

The process is iterated as long as the matrix can be reduced. Upon termination, we output the reduced matrix $C := \mathcal{N}^{2p}(M)$, for some $p \geq 1$, which is the adjacency matrix of the core K^c of K. Removing a row or column is the most basic operation of our algorithm. We will discuss it in more detail later in the paragraph *Domination test.*

Example. As mentioned before, the matrix M in the left of the Table 1 represents the simplicial complex K in the left of Figure 2. We go through the rows first, rows a and c are subsets of row b and row f is a subset of e. Removing rows a, c and f and transposing M yields the adjacency matrix $\mathcal{N}(M)$ of $\mathcal{N}(K)$ in the middle. Now, row σ_1 is a subset of σ_2 and of σ_3, and σ_5 is a subset of σ_2 and of σ_4. We remove these two rows of $\mathcal{N}(M)$ and transpose $\mathcal{N}(M)$ so as to get $\mathcal{N}^2(M)$ (the rightmost matrix of Figure 2), which corresponds to the core drawn in blue in Figure 2.

Domination test. Now we explain in more detail how to detect the rows that need to be removed. Let v be a row of M and σ_v be the associated simplex in $\mathcal{N}(K)$. If σ_v is not a maximal simplex of $\mathcal{N}(K)$, it is a proper face of some maximal simplex $\sigma_{v'}$ of $\mathcal{N}(K)$. Equivalently, the row v' of M that is associated with $\sigma_{v'}$ contains row v in the sense that the non zero elements of v appear in the same columns as the non zero elements of v'. We will say that row v is dominated by row v' and determining if a row is dominated by another one will be called the row domination test. Notice that when a row v is dominated by a row v', the same is true for the associated vertices since all the maximal simplices that contain vertex v also contain vertex v', which is the criterion to determine if v is dominated by v' (See Remark 1 in Section 2). The algorithm removes all dominated rows and therefore all dominated vertices of K.

After removing rows, the algorithm removes the columns that are no longer maximal in K, which might happen since we removed some rows. Removing a column may lead in turn to new dominated vertices and therefore new rows to be removed. When the algorithm stops, there are no rows to be removed and we have obtained the core K^c of the complex K. Note that the algorithm provides a constructive proof of Theorem 1.

Removing columns is done in very much the same way: we just exchange the roles of rows and columns.

Computing the retraction map r. The algorithm also provides a direct way to compute the retraction map r defined in Section 2. The retraction map corresponding to the strong collapses executed by the algorithm can be constructed as follows. A row r being removed in M corresponds to a dominated vertex in K and the row which *contains* r corresponds to a dominating vertex. Therefore we map the dominated vertex to the dominating vertex and compose all such maps to get the final retraction map from K to its core K^c. The final map is simplicial as well, as it is a composition of simplicial maps.

Reducing the number of domination tests. We first observe that, when one wants to determine if a row v is dominated by some other row, we don't need to test v with all other rows but with at most d of them. Indeed, at most $d + 1$ rows can intersect a given column since a simplex can have at most $d + 1$ vertices. For example, in Table 1 (Left), to check if row e (highlighted in brown) is dominated by another row, we pick the first non-zero column σ_2 (highlighted in Gray) and compare e with the non-zero entries $\{b\}$ of σ_2.

A second observation is that we don't need to test all rows and columns for domination, but only the so-called candidate rows and columns. We define a row r to be a **candidate row** for the next iteration if at least one column containing one of the non-zero elements of r has been removed in the previous column removal iteration. Similarly, by exchanging the roles of rows and columns, we define the **candidate columns**. Candidate rows and columns are the only rows or columns that need to be considered in the *domination* tests of the algorithm. Indeed, a column τ of M whose non-zero elements all belong to rows that are present from the previous *iteration* cannot be dominated by another column τ' of M, since τ was not dominated at the previous iteration and no new non-zero elements have ever been added by the algorithm. The same argument follows for the candidate rows.

We maintain two *queues*, one for the candidate columns (colQueue) and one for the candidate rows (rowQueue). These queues are implemented as First in First out (FIFO) queues. At each iteration, we *pop out* a candidate row or column from its respective queue and test whether it is dominated or not. After each successful domination test, we *push* the candidate columns or rows in their appropriate queue in preparation for the subsequent iteration. In the first iteration, we *push* all the rows in rowQueue and then alternatively use colQueue and rowQueue. Algorithm 1 gives the pseudo code of our algorithm.

Time Complexity. The most basic operation in our algorithm is to determine if a row is dominated by another given row, and similarly for columns. In our implementation, the rows (columns) of the matrix that are considered by the algorithm are stored as sorted lists. Checking if one sorted list is a subset of another sorted list can be done in time $\mathcal{O}(l)$, where l is the size of the longer list. Note that the length of a row list is at most Γ_0 where Γ_0 denotes

Algorithm 1 Core algorithm.

```
procedure CORE($M$)                ▷ Returns the matrix corresponding to the core of $K$
    $rowQueue \leftarrow$ push all rows of M (all vertices of K)
    $colQueue \leftarrow$ empty
    while rowQueue is not empty do
        $v \leftarrow pop(rowQueue)$
        $\sigma \leftarrow$ the first non-zero column of $v$
        for non-zero rows $w$ in $\sigma$ do
            if $v$ is a subset of $w$ then
                Remove $v$ from $M$
                push all non-zero columns $\tau$ of $v$ to $colQueue$ if not pushed before
                break
            end if
        end for
    end while
    while colQueue is not empty do
        $\tau \leftarrow pop(colQueue)$
        $v \leftarrow$ the first non-zero row of $\tau$
        for non-zero columns $\sigma$ in $v$ do
            if $\tau$ is subset of $\sigma$ then
                Remove $\tau$ from $M$
                push all non-zero rows $w$ of $\tau$ to $rowQueue$ if not pushed before
                break
            end if
        end for
    end while
    if rowQueue is not empty then
        GOTO 4
    end if
    return $M$                     ▷ The core consists of the remaining rows and columns
end procedure
```

an upper bound on the number of maximal simplices incident to a vertex. The length of a column list is at most $d+1$ where d is the dimension of the complex. Hence checking if a row is dominated by another row takes $\mathcal{O}(\Gamma_0)$ time and checking if a column is dominated by another column takes $\mathcal{O}(d)$ time.

At each iteration on the rows (Lines 7-13 of Algorithm 1), each row is checked against at most d other rows (since a maximal simplex has at most $d+1$ vertices), and at each iteration of the columns (Lines 18-24 of Algorithm 1), each column is checked against at most Γ_0 other columns (since a vertex can belong to at most Γ_0 maximal simplices). Since, at each iteration on the rows, we remove at least one row, the total number of iterations on the rows is at most $O(v^2)$, where v is the total number of vertices of the complex K. Similarly, at each iteration on the columns, we remove at least one column and the total number of iterations on columns is $O(m^2)$, where m is the total number of maximal simplices of the complex K. The worst-case time complexity of our algorithm is therefore $\mathcal{O}(v^2\Gamma_0 d + m^2\Gamma_0 d)$. In practice, m is much smaller than n, the total number of simplices, and Γ_0 is much smaller than Γ, the maximum number of simplices incident on a vertex. Typically Γ grows exponentially with d while Γ_0 remains almost constant as d increases. See Table 5 in [3] and related results in [4], and also the plots in Section 5.

4 Strong collapse of zigzag sequences

To be able to present our main result, we need to begin with some brief background on zigzag persistence. Readers interested in more details can refer to [6, 7, 11].

Given a **zigzag sequence** of simplicial complexes $\mathcal{Z} : \{K_1 \xrightarrow{f_1} K_2 \xleftarrow{g_2} K_3 \xrightarrow{f_3} \cdots \xrightarrow{f_{(n-1)}} K_n\}$, if we compute the homology classes of all K_is, we get the sequence $\mathcal{P}(\mathcal{Z}) : \{H_p(K_1) \xrightarrow{f_1^*} H_p(K_2) \xleftarrow{g_2^*} H_p(K_3) \xrightarrow{f_3^*} \cdots \xrightarrow{f_{(n-1)}^*} H_p(K_n)\}$. Here $H_p(-)$ denotes the homology class of dimension p with coefficients from a field $\mathbb{F}$ and $*$ denotes an induced homomorphism. $\mathcal{P}(\mathcal{Z})$ is a sequence of vector spaces connected through homomorphisms, called a **Zigzag module**. More formally, a *zigzag module* $\mathbb{V}$ is a sequence of vector spaces $\{V_1 \to V_2 \leftarrow V_3 \to \cdots \leftrightarrow V_n\}$ connected with homomorphisms $\{\to, \leftarrow\}$ between them. A zigzag module arising from a sequence of simplicial complexes captures the evolution of the topology of the sequence.

For two integers b and d, $1 \leq b \leq d \leq n$; we can define an **interval module** $\mathbb{I}[b, d]$ by assigning V_i to $\mathbb{F}$ when $i \in [b, d]$, and null spaces otherwise, the maps between any two $\mathbb{F}$ vector spaces is identity and is zero otherwise. For example $\mathbb{I}[2, 4] : \{0 \xrightarrow{0} \mathbb{F} \xleftarrow{I} \mathbb{F} \xrightarrow{I} \mathbb{F} \xleftarrow{0} 0 \xrightarrow{0} 0\}$, here $n = 6$. Any zigzag module can be *decomposed* as the direct sum of *finitely* many interval modules, which is unique upto the permutations of the interval modules [6]. The multiset of all the intervals $[b_j, d_j]$ corresponding to the interval module decomposition of any zigzag module is called a **zigzag (persistence) diagram**. The zigzag diagram completely characterizes the zigzag module, that is, there is bijective correspondence between them [6, 31].

Two different zigzag modules $\mathbb{V} : \{V_1 \to V_2 \leftarrow V_3 \to \cdots \leftrightarrow V_n\}$ and $\mathbb{W} : \{W_1 \to W_2 \leftarrow W_3 \to \cdots \leftrightarrow W_n\}$, connected through a set of homomorphisms $\phi_i : V_i \to W_i$ are **equivalent** if the ϕ_is are isomorphisms and the following diagram commutes [6, 11].

$$\begin{tikzcd}
V_1 \arrow[r] \arrow[d, "\phi_1"] & V_2 \arrow[d, "\phi_2"] & V_3 \arrow[l] \arrow[d, "\phi_3"] & \cdots & {} \arrow[r] & V_{n-1} \arrow[r] \arrow[d, "\phi_{n-1}"] & V_n \arrow[d, "\phi_n"] \\
W_1 \arrow[r] & W_2 & W_3 \arrow[l] & \cdots & {} \arrow[r] & W_{n-1} \arrow[r] & W_n
\end{tikzcd}$$

Note that the *length* of the modules and the directions of the arrows in them should be consistent. Two equivalent zigzag modules will have the same interval decomposition, therefore the same zigzag diagram.

A zigzag sequence is called a simplicial **tower** if all maps are forward. i.e. only f_is. A tower is called a **filtration** if the maps are only inclusions.

Strong collapse of the zigzag module. Given a zigzag sequence $\mathcal{Z} : \{K_1 \xrightarrow{f_1} K_2 \xleftarrow{g_2} K_3 \xrightarrow{f_3} \cdots \xrightarrow{f_{(n-1)}} K_n\}$. We define the **core sequence** $\mathcal{Z}^c$ of $\mathcal{Z}$ as $\mathcal{Z}^c : \{K_1^c \xrightarrow{f_1^c} K_2^c \xleftarrow{g_2^c} K_3^c \xrightarrow{f_3^c} \cdots \xrightarrow{f_{(n-1)}^c} K_n^c\}$. Where K_i^c is the core of K_i. The forward maps are defined as, $f_j^c := r_{j+1} f_j i_j$; and the backward maps are defined as $g_j^c := r_j g_j i_{j+1}$. The maps $i_j : K_j^c \hookrightarrow K_j$ and $r_j : K_j \to K_j^c$ are the composed inclusions and the retractions maps defined in Section 2 respectively.

▶ **Theorem 2.** *Zigzag modules $\mathcal{P}(\mathcal{Z})$ and $\mathcal{P}(\mathcal{Z}^c)$ are equivalent.*

Proof. Consider the following diagram

$$\begin{tikzcd}
K_1 \arrow[r, "f_1"] \arrow[d, "r_1"] & K_2 \arrow[d, "r_2"] & K_3 \arrow[l, "g_2"'] \arrow[d, "r_3"] & \cdots \arrow[r] & K_{n-1} \arrow[r, "f_{n-1}"] \arrow[d, "r_{n-1}"] & K_n \arrow[d, "r_n"] \\
K_1^c \arrow[r, "f_1^c"] & K_2^c & K_3^c \arrow[l, "g_2^c"'] & \cdots \arrow[r] & K_{n-1}^c \arrow[r, "f_{n-1}^c"] & K_n^c
\end{tikzcd}$$

and the associated diagram after computing the p-th homology groups

$$\begin{tikzcd}
H_p(K_1) \arrow[r, "f_1^*"] \arrow[d, "r_1^*"] & H_p(K_2) \arrow[d, "r_2^*"] & H_p(K_3) \arrow[l, "g_2^*"'] \arrow[d, "r_3^*"] & \cdots \arrow[r] & H_p(K_{n-1}) \arrow[r, "f_{n-1}^*"] \arrow[d, "r_{n-1}^*"] & H_p(K_n) \arrow[d, "r_n^*"] \\
H_p(K_1^c) \arrow[r, "(f_1^c)^*"] & H_p(K_2^c) & H_p(K_3^c) \arrow[l, "(g_2^c)^*"'] & \cdots \arrow[r] & H_p(K_{n-1}^c) \arrow[r, "(f_{n-1}^c)^*"] & H_p(K_n^c)
\end{tikzcd}$$

Since there exists a strong deformation retract between r_j and i_j, the induced homomorphisms r_j^* and i_j^* are isomorphisms [17, Corollary 2.11]. Also, $f_j^c r_j = r_{j+1} f_j i_j r_j$ is contiguous to $r_{j+1} f_j$, since $i_j r_j$ is contiguous to the identity on K_j and contiguity is preserved under composition, see [2, Proposition 2.9] and similarly $g_j^c r_{j+1}$ is contiguous to $r_j g_j$. Now, since contiguous maps are homotopic at the level of geometric realization and homotopic maps induce the same homomorphism, we have $(f_j^c r_j)^* = (r_{j+1} f_j)^*$ and thus $(f_j^c)^* r_j^* = r_{j+1}^* f_j^*$ and similarly $(g_j^c)^* r_{j+1}^* = r_j^* g_j^*$, see [17, Proposition (1) page 111]. Therefore all the squares in the lower diagram commute and the set of maps r_j^*s are isomorphisms, therefore $\mathcal{P}(\mathcal{Z})$ and $\mathcal{P}(\mathcal{Z}^c)$ are *equivalent* and hence their zigzag diagrams are identical. ◀

In fact, using the more general notion of quiver representation [11], this result follows for the multidimensional persistence as well.

5 Computational experiments

For each data set, we consider as the input sequence a nested sequence (filtration) of Vietoris-Rips (VR) complexes associated with a set of increasing values of the scale parameter (called snapshots). The snapshots are specific values of the scale parameter at which we choose to strong collapse the complex. The choice of snapshots strongly dictates the performance and the quality of computed PD. Sparse snapshots will lead to faster computation but to coarser PD where the points of persistence less than the interval between two snapshots have been removed. On the other hand, choosing denser snapshots will lead to a comparatively slower algorithm but will provide more refined PD. We first *independently* strong collapse all these complexes, then assemble the resulting individual *cores* using the induced simplicial maps introduced in Section 4. The resulting new sequence with induced simplicial maps between the collapsed complexes is usually a simplicial tower we call the *core tower.* We then convert the core tower into an equivalent filtration using the Sophia software [26], which implements the algorithm described by Kerber and Schreiber in [18]. Finally, we run the persistence algorithm of the Gudhi library [16] to obtain the persistence diagram (PD) of the equivalent filtration. By the results of Section 4, the obtained PD is the same as the PD of the initial sequence.

The total time to compute the PD of the core sequence is the sum of three terms: 1. the *maximum* time taken to collapse all the individual complexes (assuming they are computed in parallel), 2. the time taken to assemble the individual *cores* to form the core tower, 3. the time to compute the persistent diagram of the core tower. Table 2 summarises the results of the experiments. In both cases, the original filtration and the core tower, we use Gudhi through Sophia using the command <./sophia -cgudhi inputTowerFile outputPDFile>. When we use the -cgudhi option, Sophia reports two computation times. The first one is the total time

Table 2 The rows are, from top to down: dataset $\mathcal{X}$, number of snapshots (snp), total number of simplices in the original filtration (Flt) in millions, number of simplices in the collapsed tower (Twr), total number of simplices in the equivalent filtration (EqF), ratio of Flt and EqF (Flt/EqF) in thousands, PD computation time for the original filration (PDF), maximum collapse time (MCT), assembly time (AT), PD computation time of the tower (PDT), sum MCT+AT+PDT (Total), ratio of PDF and Total (PDF/Total). All times are noted in seconds. For the first three datasets, we sampled points randomly from the initial datasets and averaged the results over five trials.

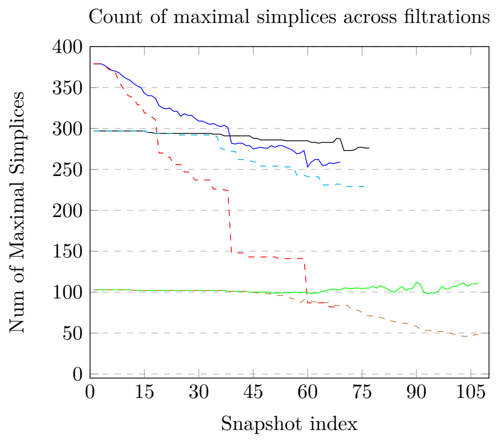
$\mathcal{X}$	1-sphere	2-Annulus	dragon	netw-sc	senate	eleg
Snp	80	80	46	69	107	77
Flt(10^6)	0.12	13.91	7.96	22.35	2.56	1.18
Twr	54	252	1,641	380	104	298
EqF	573	1,954	8,437	957	270	431
Flt/EqF(10^3)	0.21	7.12	0.94	23.35	9.48	2.74
PDF	0.65	174.18	69.92	243.86	24.92	10.87
MCT	0.005	0.022	0.065	0.009	0.003	0.002
AT	0.045	0.136	0.408	0.078	0.06	0.157
PDT	0.01	0.02	0.08	0.01	0.005	0.006
Total	0.060	0.178	0.553	0.097	0.068	0.165
PDF/Total	10.8	978.5	126.4	2514.0	366.5	65.9

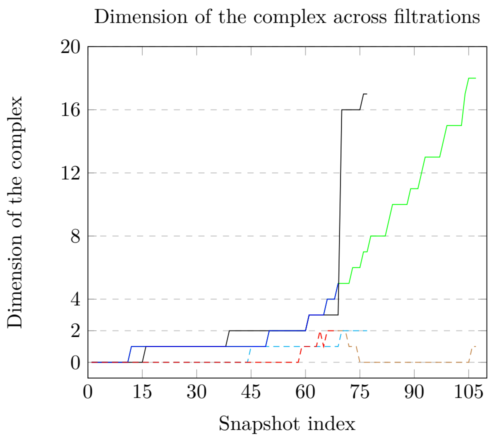
taken by Sophia which includes (1) reading the tower, (2) transforming it to a filtration and (3) computing PD using Gudhi. The second reported time is just the time taken by Gudhi to compute PD. In our comparisons, we just report the time taken by Gudhi for the original filtration, while, for the core tower, we report the total time taken by Sophia.

The dataset of the first column (1-sphere) of Table 2 consists of 100 random points sampled from a unit circle in dimension 2. The dataset of the second column (2-Annulus) consists of 150 random points sampled from a two dimension annulus of radii $\{0.6, 1\}$. For all the other experiments, we use datasets from a publicly available repository [10]. These datasets have been previously used to benchmark different publicly available software computing PH [23]. For the third experiment (*dragon*), we randomly picked 150 points from the 2000 points of the dataset **drag 2** of [10]. The fourth and fifth column respectively correspond to the dataset **netw-sc** and **senate** of [10], here we used the distance matrix. The sixth column corresponds to the dataset **eleg** of [10], and here again we used the distance matrix. The first three datasets are point sets in Euclidean space. For the other three, the distance matrices of the datasets were available at [10]. The [initial value, increment, final value] of the scale parameter are $[0.1, 0.005, 0.5]$, $[0.1, 0.005, 0.5]$, $[0, 0.001, 0.046]$, $[0.1, 0.05, 3.5]$, $[0, 0.001, 0.107]$ and $[0, 0.001, 0.077]$ for the examples in Table 2 (from left to right). The filtration value of a simplex is the value of the snapshot at which it first appeared. For more detail about the datasets and the computation of the distance matrices of the last three datasets please refer to [23].

The plots below count the maximal simplices and the dimensions of the complexes across the filtration (in solid) and the collapsed tower (as dashed). Blue and red correspond respectively to the filtration and the collapsed tower of the data **netw-sc**. Similarly green and brown correspond respectively to the filtration and the collapsed tower of the data **senate**. Finally, **black** and cyan correspond to the filtration and the collapsed tower of the data

eleg respectively. We can observe that in all cases the number of maximal simplices never increases. Also they are far fewer in number compared to the total number of simplices. Observe that for the uncollapsed filtrations blue, green and **black**, the dimension of the complexes increases quite rapidly with the snapshot index. Another key fact to observe is that the dimension of the complexes in the corresponding core tower are much smaller than their counterparts in the filtration. This has a huge effect on the performances since the total number of simplices depends exponentially on the dimension.





Noticeably, in our experiments, the computing time of our approach is reduced by 1 to 3 orders of magnitude, and the gain increases with the size of the filtration. A similar reduction of 2 to 4 orders of magnitude is achieved for the number of simplices. Observations from the plots combined with the experimental results of Table 2 clearly indicate that our method is extremely fast and memory efficient.

The implementation of the Core algorithm 1 bench-marked here is coded in C++ and will be available as an open-source package of the next release of the Gudhi library [16]. The code was compiled using the compiler <clang-900.0.38> and all computations were performed on <2.8 GHz Intel Core i5> machine with 16 GB of available RAM.

The experiments above are limited to filtrations of VR-complexes, by far the most commonly used type of sequences in Topological Data Analysis. We intend to experiment on Zigzag sequences in future work.

References

1 M. Adamaszek and J. Stacho. Complexity of simplicial homology and independence complexes of chordal graphs. *Computational Geometry: Theory and Applications*, 57:8–18, 2016.

2 J. A. Barmak and E. G. Minian. Strong homotopy types, nerves and collapses. *Discrete and Computational Geometry*, 47:301–328, 2012.

3 Jean-Daniel Boissonnat, C. S. Karthik, and Sébastien Tavenas. Building efficient and compact data structures for simplicial complexes. *Algorithmica*, 79:530–567, 2017.

4 Jean-Daniel Boissonnat and Karthik C. S. An efficient representation for filtrations of simplicial complexes. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2017.

5 M. Botnan and G Spreemann. Approximating persistent homology in euclidean space through collapses. *In: Applicable Algebra in Engineering, Communication and Computing*, 26:73–101, 2014.

6 Gunnar Carlsson and Vin de Silva. Zigzag persistence. *Found Comput Math*, 10, 2010.

7 Gunnar Carlsson, Vin de Silva, and Dmitriy Morozov. Zigzag persistent homology and real-valued functions. *SOCG*, pages 247–256, 2009.

8 F. Chazal and S. Oudot. Towards persistence-based reconstruction in euclidean spaces. *SOCG*, 2008.

9 Aruni Choudhary, Michael Kerber, and Sharath Raghvendra:. Polynomial-sized topological approximations using the permutahedron. In *32nd International Symposium on Computational Geometry, SoCG 2016, June 14-18, 2016, Boston, MA, USA*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

10 Datasets. URL: `https://github.com/n-otter/PH-roadmap/''`.

11 Harm Derksen and Jerzy Weyman. Quiver representations. *Notices of the American Mathematical Society*, 52(2):200–206, February 2005.

12 Tamal Dey, Dayu Shi, and Yusu Wang. *SimBa: An efficient tool for approximating Rips-filtration persistence via Simplicial Batch-collapse.* In European Symp. on Algorithms (ESA), pages 35:1–35:16, 2016.

13 Dionysus. URL: `http://www.mrzv.org/software/dionysus/`.

14 P. Dłotko and H. Wagner. Simplification of complexes for persistent homology computations,. *Homology, Homotopy and Applications*, 16:49–63, 2014.

15 François Le Gall. Powers of tensors and fast matrix multiplication. *ISSAC '*, 14:296–303, 2014.

16 Gudhi: Geometry understanding in higher dimensions. URL: `http://gudhi.gforge.inria.fr/`.

17 A. Hatcher. *Algebraic Topology.* Univ. Press Cambridge, 2001.

18 Michael Kerber and Hannah Schreiber:. Barcodes of towers and a streaming algorithm for persistent homology. *33rd International Symposium on Computational Geometry*, 2017. `arXiv:1701.02208`.

19 Michael Kerber and R. Sharathkumar. Approximate cech complex in low and high dimensions. In *Algorithms and Computation*, pages 666–676. by Leizhen Cai, Siu-Wing Cheng, and Tak-Wah Lam. Vol. 8283. Lecture Notes in Computer Science, 2013.

20 Nikola Milosavljevic, Dmitriy Morozov, and Primoz Skraba. Zigzag persistent homology in matrix multiplication time. In *Symposium on Computational Geometry (SoCG)*, 2011.

21 K. Mischaikow and V. Nanda. Morse theory for filtrations and efficient computation of persistent homology. *DCG*, 50:330–353, September 2013.

22 J. Munkres. *Elements of Algebraic Topology.* Perseus Publishing, 1984.

23 N. Otter, M. Porter, U. Tillmann, P. Grindrod, and H. Harrington. A roadmap for the computation of persistent homology. *EPJ Data Science, Springer Nature*, page 6:17, 2017.

24 Ripser. URL: `https://github.com/Ripser/ripser`.

25 Donald Sheehy. Linear-size approximations to the vietoris–rips filtration. *Discrete and Computational Geometry*, 49:778–796, 2013.

26 Sophia. URL: `https://bitbucket.org/schreiberh/sophia/`.

27 J.Reininghausc U. Bauer, M. Kerber and Hagner:. Phat – persistent homology algorithms toolbox. *Journal of Symbolic Computation*, 78, 2017.

28 J. H. C Whitehead. Simplicial spaces nuclei and m-groups. *Proc. London Math. Soc*, 45:243–327, 1939.

29 A. C. Wilkerson, H. Chintakunta, and H. Krim. Computing persistent features in big data: A distributed dimension reduction approach. *ICASSP - Proceedings*, pages 11–15, 2014.

30 A. C. Wilkerson, T. J. Moore, and and A. H. Krim A. Swami. *Simplifying the homology of networks via strong collapses.* ICASSP - Proceedings, 2013.

31 A. Zomorodian and G. Carlsson. Computing persistent homology. *Discrete Comput. Geom*, 33:249–274, 2005.

On the Complexity of the (Approximate) Nearest Colored Node Problem

Maximilian Probst
BARC, University of Copenhagen, Universitetsparken 5, Copenhagen 2100, Denmark
probst@di.ku.dk
https://orcid.org/0000-0003-3522-156X

Abstract

Given a graph $G = (V, E)$ where each vertex is assigned a color from the set $C = \{c_1, c_2, .., c_\sigma\}$. In the (approximate) nearest colored node problem, we want to query, given $v \in V$ and $c \in C$, for the (approximate) distance $\widehat{\mathbf{dist}}(v, c)$ from v to the nearest node of color c. For any integer $1 \leq k \leq \log n$, we present a Color Distance Oracle (also often referred to as Vertex-label Distance Oracle) of stretch $4k - 5$ using space $O(kn\sigma^{1/k})$ and query time $O(\log k)$. This improves the query time from $O(k)$ to $O(\log k)$ over the best known Color Distance Oracle by Chechik [6].

We then prove a lower bound in the cell probe model showing that even for unweighted undirected paths any static data structure that uses space S requires at least $\Omega\left(\log \frac{\log \sigma}{\log(S/n)+\log\log n}\right)$ query time to give a distance estimate of stretch $O(\text{polylog}(n))$. This implies for the important case when $\sigma = \Theta(n^\epsilon)$ for some constant $0 < \epsilon < 1$, that our Color Distance Oracle has asymptotically optimal query time in regard to k, and that recent Color Distance Oracles for trees [23] and planar graphs [16] achieve asymptotically optimal query time in regard to n.

We also investigate the setting where the data structure additionally has to support color-reassignments. We present the first Color Distance Oracle that achieves query times matching our lower bound from the static setting for large stretch yielding an exponential improvement over the best known query time [7]. Finally, we give new conditional lower bounds proving the hardness of answering queries if edge insertions and deletion are allowed that strictly improve over recent bounds in time and generality.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms, Theory of computation → Graph algorithms analysis, Theory of computation → Cell probe models and lower bounds

Keywords and phrases Nearest Colored Node, Distance Oracles, Cell-probe lower bounds

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.68

Related Version A full version of the paper is available at `https://arxiv.org/abs/1807.03721`.

Funding Supported by Basic Algorithms Research Copenhagen (BARC), supported by Thorup's Investigator Grant from the Villum Foundation under Grant No. 16582.

Acknowledgements I want to thank Christian Wulff-Nilsen for inspiring the research on this problem and the guidance during the project.

1 Introduction

In the *static* nearest colored node problem, we are given a graph $G = (V, E)$ and a color set $C = \{c_1, c_2, .., c_\sigma\}$ with a function $c : V \to C$ mapping each vertex to a color in C and we want to compute and store the distance $\mathbf{dist}_G(u, c)$ denoting the distance from each vertex

© Maximilian Probst;
licensed under Creative Commons License CC-BY

Table 1 Best upper and lower bounds for static Color Distance Oracles. N denotes the weight of the heaviest edge in the graph. The w in the second column refers to the word size and we assume in this article that $w = \Theta(\log n)$. N refers to the heaviest edge weight in the graph.

Graph Family	Approximation	Space	Query Time	Ref
Unweighted undirected path	$O(\text{polylog}(n))$	S	$\Omega\left(\log \frac{\log \sigma}{\log(S/n)+\log\log n}\right)$	**New**
Tree	Exact	$O(n)$	$O\left(\log \frac{\log \sigma}{\log w}\right)$	[23]
Planar graphs	$1+\epsilon$	$O(n \log n)$	$O(\log\log n)$	[16]
Planar digraphs	$1+\epsilon$	$O(n \log n)$	$O(\log\log n$ $\log\log(nN))$	[16]
General graphs	$4k-5$	$O(kn\sigma^{1/k})$	$O(\log k)$	**New**
General graphs	$8(1+\epsilon)k$	$O(kn^{1+1/k}\log n)$	$O\left(\log \frac{\log \sigma}{\log w}\right)$	**New**

$u \in V$ to the nearest vertex v of color $c \in C$ or more formally $\mathbf{dist}_G(u,c) = \mathbf{min}_{v \in V_c}\mathbf{dist}(u,v)$ where V_c denote the set of c-colored vertices. Clearly, if the color set C is of small cardinality, computing distances and storing a distance matrix is an efficient measure but for large σ, i.e. $\sigma = \Omega(n^\epsilon)$ for some $\epsilon > 0$, this solution becomes impractical for many important applications. The problem has various applications in navigation, routing and document processing, often in connection to locating resources or facilities, e.g. the nearest gas station, quickly.

In this article, we present a new data structure that reports for any given $v \in V, c \in C$ a distance estimate $\widehat{\mathbf{dist}}(v,c)$ in $O(\log k)$ time such that $\mathbf{dist}(v,c) \leq \widehat{\mathbf{dist}}(v,c) \leq (4k-5)\mathbf{dist}(v,c)$ using $O(kn\sigma^{1/k})$ space for every positive integer k, improving on $O(k)$ query time of the previous data structure with same space consumption and stretch factor [6]. We refer to the data structure as *Color Distance Oracle*. In other literature, the problem is also studied as the vertex-to-label problem and the data structure is referred to as Vertex-label Distance Oracle [13][6]. The term Distance Oracle originates from the classic Thorup-Zwick Distance Oracle [22] that reports distance estimates for each pair of vertices. Thorup, Zwick and Roddity [20] also generalized Distance Oracles by introducing source-restricted Distance Oracles where only distances in $S \times V$ can be queried for some subset $S \subseteq V$ and that uses space $O(n|S|^{1/k})$ for stretch $2k-1$. It is tempting to approach the nearest colored node problem by using an auxiliary vertex for each color $c \in C$ linking it to all c-colored vertices with a zero-weight edge and let the auxiliary vertices define the set S. Unfortunately, this might decrease some vertex-to-color distances by creating “portals” through the auxiliary vertices. Instead the underlying sampling techniques can, together with a more advanced analysis stemming from Compact Routing Schemes [21], be used to construct efficient and correct Color Distance Oracles. Color Distance Oracles can also be seen as a generalization of (source-restricted) Distance Oracles as we can choose a color set of size $|S|$ and assign each vertex in S a unique color.

We first present a Color Distance Oracle of space $O(kn\sigma^{1/k})$ that reports in $O(\log k)$ time distance estimate of stretch at most $(4k-5)$. Our Color Distance Oracle matches space and stretch of the best data structure by Chechik [6] and improves the query time from $O(k)$ to $O(\log k)$. This is in fact achieved by combining Chechiks result with a well-known technique by Wulff-Nilsen [25] for general distance oracles. Our contribution is to simplify the technique of Wulff-Nilsen by observing that a Range Minimum Query (RMQ) data structure can be used to replace his tailor-made data structure even more efficiently resulting in a concise and simple algorithm; and to generalize the proof technique of Wulff-Nilsen. Recently, Chechik has also shown that classic Distance Oracles can be implemented with constant query time [8][7] and it is natural to ask whether this improvement carries over to

Table 2 Best upper and lower bounds for Color Distance Oracles supporting color-reassignments. Here N denote the heaviest edge weight in the graph. The lower bound from [12] applies to update or query time for every $\epsilon > 0$. More precisely, any algorithm with update time $\tilde{o}(n^{1-\epsilon})$ and $\tilde{o}(n^{2-\epsilon})$ would refute the OMv-conjecture.

Graph Family	**Approx-imation**	**Update Time**	**Query Time**	**Ref**
Unweighted undirected path	$O(\text{polylog}(n))$	$O(\text{polylog}(n))$	$\Omega\left(\log \frac{\log \sigma}{\log(S/n)+\log\log n}\right)$	**New**
Unweighted tree	exact	$O(\text{polylog}(n))$	$\Omega\left(\frac{\log(n)}{\log\log(n)}\right)$	[11]
Tree	exact	$O(\log n)$	$O(\log n)$	[11]
	exact	$O(\log^{1+\epsilon} n)$	$O\left(\frac{\log(n)}{\log\log(n)}\right)$	[11]
Planar graphs	$1+\epsilon$	$O(\epsilon^{-1}\log(n) \log\log(n))$	$O(\epsilon^{-1}\log(n)\log(nN) \log\log(n))$	[14]
	$1+\epsilon$	$O\left(\epsilon^{-1}\frac{\log^2(\epsilon^{-1}n)}{\log\log(n)}\right)$	$O(\epsilon^{-1}\log^{1.51}(\epsilon^{-1}n))$	[14]
Planar digraphs	$1+\epsilon$	$O(\epsilon^{-1}\log(n) \log\log(nN))$	$O(\epsilon^{-1}\log^3(n)\log(nN))$	[14]
General graphs	< 3	$\Omega(n^{1-\epsilon})$	$\Omega(n^{2-\epsilon})$	[12]
	$4k-5$	$O(kn^{1/k}\log^{1-1/k} n \log\log n)$	$O(k)$	[6]
	$8(1+\epsilon)k$	$O\left(\epsilon^{-1}kn^{1/k}\log\log n\right)$	$O(\log\log n)$	**New**
	$8(1+\epsilon)k$	$O(\log\log n)$	$O\left(\epsilon^{-1}kn^{1/k}\log\log n\right)$	**New**

Color Distance Oracles. Our new lower bound rules out such an improvement and shows that our Color Distance Oracle has essentially tight query time when $\sigma = \Theta(n^\epsilon)$ for constant $0 < \epsilon < 1$, as the lower bound then simplifies to $\Omega(\log(\epsilon k))$ for $S = n^{1+1/k}$ for all values of $k = O(\text{polylog}(n))$. Our result extends to prove asymptotic optimality in query time in regard to n for the best known Color Distance Oracles for trees [11][23] and planar graphs [16] even for data structures with higher stretch $k = O(\text{polylog}(n))$. This lower bound, that is our main contribution, is thus a significant step in understanding Color Distance Oracles and their limitations. An overview over the best upper bounds for different graph families and our lower bound is given in table 1.

We also present a new Color Distance Oracle for the setting where the data structure needs to handle color-reassignments, i.e. updates in which a vertex $v \in V$ is assigned a new color $c \in C$ such that afterwards $c(v) = c$. Our Color Distance Oracle is conceptually simple, building on some recent results in Ramsey theory[2] and can be constructed deterministically. It strictly improves on query and update time for any approximation factor $k = \Omega(\frac{\log n}{\log\log n})$ and dominates existing data structures in query time for $k = \Omega(\log\log n)$. We are also able to show an elegant trade-off between query and update time that was unknown before. For $k = \Omega(\frac{\log n}{\log\log n})$ our data structure requires $\tilde{O}(n)$ space and updates only take polylogarithmic time. Therefore our static lower bound extends to this setting as we can start with an uncolored graph and color vertices in n updates. This implies that our query time is tight with regard to n. Achieving query time $O(\log\log n)$ is rather surprising given that queries for exact distances take $\Omega(\frac{\log n}{\log\log n})$ time even on unweighted balanced trees and that for the static setting approximation doesn't admit any query time improvements. An overview over upper bounds and lower bounds for the color-reassignment setting is given in table 2.

Finally, we prove that the *dynamic* version of the problem, allowing edge insertions and deletions, cannot process updates in time $\tilde{\tilde{o}}(\sigma)$[1] and queries in time $\tilde{\tilde{o}}(n/\sigma)$ even on unweighted path graphs for queries that ask to report given a fixed source $s \in V$ whether there is a vertex of color c in the same component, unless Online Matrix Multiplication(OMv) has a truly subcubic time algorithm. We then show that even update time $\tilde{\tilde{o}}(\sigma)$ and query time $\tilde{\tilde{o}}(n)$ is not possible if we have directed general graph or ask for distance queries of approximation factor $< 5/3$. Combined they strictly improve in generality and query time over a recent lower bound by Gawrychowski et al. [11] showing that for weighted trees, query and update time $\tilde{\tilde{o}}(\sqrt{n})$ where $\sigma = \Theta(\sqrt{n})$ would imply a truly subcubic solution to tripartite APSP. Our reduction implies an interesting connection to Pagh's problem and the lower bound is in fact obtained by adapting the reduction from Pagh's problem in [12].

2 Preliminaries

We denote by $\mathbf{dist}(u, v)$ for $u, v \in V$ the shortest-path distance from u to v and by $\mathbf{dist}(u, c)$ with $u \in V, c \in C$ the shortest-path distance between u and the nearest vertex of color c. When the context is clear, we often only refer to the nearest colored vertex instead of the nearest vertex of color c and let c denote the color under consideration. We let $v = \textsc{Nearest}(u, c)$ denote the nearest vertex of color c to u, i.e. $\mathbf{dist}(u, c) = \mathbf{dist}(u, \textsc{Nearest}(u, c))$. We also make use of the following data structures.

Predecessor Search Problem. In the predecessor search problem, we are given a universe $U = \{0, .., m-1\} = [m]$ and a subset $S \subseteq U$ of size $n = |S|$. Given an element $x \in U$, we ask for the largest element in S that is smaller than x or more formally, we ask for the predecessor of x, $\textsc{Pred}(x) = \max\{y \in S | y < x\}$. We let the successor of x be $\textsc{Succ}(x) = \min\{y \in S | y > x\}$. Pătraşcu and Thorup present in [18] a predecessor search data structures that solves queries and updates (insertions and deletions into/from S) in $O(\log\log n)$ time and linear space and give a lower bound of $\Omega(\log\log n / \log\log\log n)$ if $m \leq \text{poly}(n)$ and only almost linear-space is given.

Range Minimum Query (RMQ). A RMQ is a structure augmenting an array $A[1..n]$ answering queries of the form $\textsc{Rmq}(i, j) = \min_{k \in [i,j]} A[k]$ by returning the index of field with the minimal value, for any $1 \leq i \leq j \leq n$. RMQ can be solved with $O(n)$ preprocessing time, taking $O(n)$ space and $O(1)$ query time [5][10].

Least Common Ancestor (LCA). The LCA problem is the problem of finding the least common ancestor in T, which we denote $\textsc{Lca}(x, y)$ T, of any two nodes $x, y \in \mathcal{V}(T)$. The LCA problem can be reduced to RMQ and can therefore be implemented within the same bounds.

Hash table. Given a universe $U = [m]$ and a (dynamic) subset $S \in U$, with $n = |S|$, we can query for any $x \in U$ whether x is in S. In [9], a data structure is presented that can run deterministic queries in constant time and updates of the set S, i.e. insertions and deletions, in constant amortized time.

[1] We use the $\tilde{\tilde{o}}(f(n))$-notation as introduced by Henzinger et al. [12] to denote that the running time is in $O(f(n)^{1-\epsilon})$ for some $\epsilon > 0$. For multiple parameters we let $\tilde{\tilde{o}}(n_1 n_2 n_3)$ be equivalent to $O(n_1^{1-\epsilon} n_2 n_3 + n_1 n_2^{1-\epsilon} n_3 + n_1 n_2 n_3^{1-\epsilon})$ for some $\epsilon > 0$.

3 Static Color Distance Oracle

We construct our Color Distance Oracle as in [6] with the classic techniques by Thorup and Zwick [22]: For a given positive integer k, we construct the vertex sets $V = A_0 \supseteq A_1 \supseteq A_2 \supseteq \cdots \supseteq A_{k-1}$, where A_i is obtained by sampling each vertex in A_{i-1} with probability $\sigma^{-1/k}$, for $1 \leq i \leq k-1$, and define the set $A_k = \emptyset$. For each vertex v in V, we store for each set A_i with $0 \leq i \leq k-1$ the closest neighbour in A_i denoted by $p_i(v)$, where we break ties arbitrarily. For every $v \in V$, we define $\Delta_i(v) = \mathbf{dist}(v, p_{i+1}(v)) - \mathbf{dist}(v, p_i)$, for $0 \leq i < k-1$. We then store all such distances in a consecutive array $P_v[0..k-2]$ with $P_v[i] = \Delta_i(v)$ for all i and augment P_v by a RMQ-structure that returns the maximum value in a subarray. We denote a query on the RMQ structure over P_v in the range a to b by $\textsc{Rmq}_v(a,b)$ for any $0 \leq a \leq b < k-1$. We define a *bunch* $B(v)$ for every vertex v in V as follows

$$B(v) = \bigcup_{i=0}^{k-1} \{u \in A_i \setminus A_{i+1} | \mathbf{dist}(v,u) < \mathbf{dist}(v, p_{i+1}(v))\}$$

and construct for every color $c \in C$ a bunch $B(c) = \bigcup_{v \in V_c} B(v)$. With each $B(c)$, we store in a hash table the vertices $v \in B(c)$ and associate with their key the distance $\mathbf{dist}(v,c)$. This completes our construction. The following lemma bounds space and construction time, but we defer the proof to the full version since it only differs by bounding the space of the RMQ data structures from the proof in [6].

▶ **Lemma 1.** *We use at most space $O(kn\sigma^{1/k})$ to represent the Color Distance Oracle and construction time $O(m\sigma)$.*

Note that the construction cost can be slightly improved for $\sigma > n^{k/(2k-1)}$ using the construction of Hermelin et al. [13] but our query time improvement doesn't carry over to their construction. We give the following query algorithm for color $c \in C$ and vertex $v \in V$:

Listing 1 Query(v,c)

```
$lower\_bound \leftarrow 0$
$upper\_bound \leftarrow k-1$
While $upper\_bound \neq lower\_bound$
Do
    $i \leftarrow \lceil (lower\_bound + upper\_bound)/2 \rceil$
    // Compute index $j$ such that $\Delta_j(v) = \max_{a \in \{lower\_bound,..,i-1\}} \Delta_a(v)$
    $j \leftarrow \textsc{Rmq}_v(lower\_bound, i-1)$
    If $p_j(v) \notin B(c)$
    Then
        $lower\_bound \leftarrow i$
    Else
        $upper\_bound \leftarrow j$
    End
End
Return $p_{lower\_bound}(v)$
```

We claim that the query procedure returns a colored vertex $w_c = p_{lower\_bound}(v)$ whose distance to v is $\mathbf{dist}(v,c) \leq \mathbf{dist}(v, w_c) \leq (4k-5)\mathbf{dist}(v,c)$. For the rest of the section, we let $w_{best} = \textsc{Nearest}(v,c)$.

As in [25], we let $\mathcal{I}$ be the sequence $0,..,k-1$. We call an index $j \in \mathcal{I}$, *(v,c)-terminal* if $p_j(v) \in B(c)$. We say that a subsequence $i_1,..,i_2$ of $\mathcal{I}$ is *(v,c)-feasible* if (1) $\mathbf{dist}(v, p_{i_1}(v)) \leq 2i_1\mathbf{dist}(v,c)$, and (2) i_2 is *(v,c)-terminal*. Using these definitions we are ready to prove our claim in the ensuing two lemmas.

▶ **Lemma 2.** *Let $i_1,..,i_2 \subseteq \mathcal{I}$, with $|\mathcal{I}| > 1$, be (v,c)-feasible and let $i = \lceil (i_1+i_2)/2 \rceil$. Let $\mathcal{I}'$ be the sequence $i_1,..,i-1$. Let j be the index in $\mathcal{I}'$, that maximizes $\Delta_j(v)$. Then if $j \notin B(c)$ the subsequence $i,..,i_2$ is (v,c)-feasible. Otherwise, the subsequence $i_1,..,j$ is (v,c)-feasible. The obtained subsequence is of size at most $\frac{2}{3}\mathcal{I}$.*

Proof. If $p_j(v) \in B(c)$, then j is (v,c)-*terminal*. Hence $i_1,..j$ is (v,c)-*feasible*. As $j \leq \lceil (i_1+i_2)/2 \rceil - 1 < (i_1+i_2)/2$ and $|\mathcal{I}| > 1$ the subsequence is of size at most $\frac{1}{2}\mathcal{I}$. Now, consider the case where $p_j(v) \notin B(c)$. Then

$$\mathbf{dist}(w_{best}, p_{j+1}(w_{best})) < \mathbf{dist}(w_{best}, p_j(v))$$

We can now employ the analysis from [21]:

$$\begin{aligned}\mathbf{dist}(v, p_{j+1}(v)) &\leq \mathbf{dist}(v, p_{j+1}(w_{best})) \leq \mathbf{dist}(w_{best}, v) + \mathbf{dist}(w_{best}, p_{j+1}(w_{best}))\\ &< \mathbf{dist}(w_{best}, v) + \mathbf{dist}(w_{best}, p_j(v)) \leq 2\mathbf{dist}(w_{best}, v) + \mathbf{dist}(v, p_j(v)) \quad (1)\end{aligned}$$

Therefore $\Delta_j(v) = \mathbf{dist}(v, p_{j+1}(v)) - \mathbf{dist}(v, p_j(v)) \leq 2\mathbf{dist}(w_{best}, v)$. Since $i_1,..,i_2$ is (v,c)-*feasible*, we have $\mathbf{dist}(v, p_{i_1}(v)) \leq 2i_1\mathbf{dist}(v,c)$. By choice of j,

$$\begin{aligned}\mathbf{dist}(v, p_i(v)) &= 2i_1\mathbf{dist}(v, w_{best}) + \sum_{j' \in \mathcal{I}'} \Delta_{j'}(v)\\ &\leq 2i_1\mathbf{dist}(v, w_{best}) + |\mathcal{I}'| \max_{j' \in \mathcal{I}'} \Delta_{j'}(v)\\ &= 2i_1\mathbf{dist}(v, w_{best}) + (i - i_1)\Delta_j(v)\\ &= 2i\mathbf{dist}(v, w_{best}) \quad (2)\end{aligned}$$

As i_2 is (v,c)-*terminal*, we therefore get that $i,..,i_2$ is (v,c)-*feasible*. It is now easy to see that by choice of i and as $|\mathcal{I}| > 1$ the derived sequence is smaller $\frac{2}{3}|\mathcal{I}|$. ◀

▶ **Lemma 3.** *The algorithm given in procedure* Query(v,c) *reports a distance estimate with stretch at most $(4k-5)$ in time $O(\log k)$.*

Proof. By lemma 2 the number of potential indices reduces by factor $\frac{2}{3}$ by every iteration of the loop. Therefore, we have at most $log_{\frac{3}{2}} k$ iterations. As querying the RMQ data structure takes constant time the overall running time is $O(\log k)$.

To show stretch of at most $4k-5$, we observe that the final sequence has only a single index $j \leq k-1$, and as the sequence is still (v,c)-*feasible*, we get by property (1) that $\mathbf{dist}(v, p_j(v)) \leq 2j\mathbf{dist}(v, w_{best}) \leq 2(k-1)\mathbf{dist}(v, w_{best})$. By property (2), we get that $p_j(v) \in B(c)$ and we can bound

$$\begin{aligned}\mathbf{dist}(p_j(v), c) &= \mathbf{dist}(p_j(v), w_{best}) \leq \mathbf{dist}(v, w_{best}) + \mathbf{dist}(v, p_j(v))\\ &\leq \mathbf{dist}(v, w_{best}) + 2(k-1)\mathbf{dist}(v, w_{best}) \quad (3)\end{aligned}$$

giving an overall distance of $\mathbf{dist}(v, p_j(v)) + \mathbf{dist}(p_j(v), c) \leq (4k-3)\mathbf{dist}(v, w_{best})$. This can be slightly improved to stretch $4k-5$ by using the technique from [21](Lemma A.2) even without changing the overall approach. We only have to adapt property (1) in the definition of (v,c)-*feasible* sequences $i_1,..,i_2$ to $\mathbf{dist}(v, p_{i_1}(v)) \leq (2i_1 - 1)\mathbf{dist}(v,c)$, and adapt lemma 2 to directly get the improvement. ◀

We point out that the technique presented to query the Color Distance Oracle extends to Compact Routing Schemes as described by Thorup and Zwick [21] such that paths can be computed in $O(\log k)$ time. By using a recently devised succinct RMQ structure that is only allowed to query intervals where both indices are multiples of $\log k$ by Tsur [23], we can adapt our approach to reduce the intervals as described in lemma 2 down to size $O(\log k)$ and then test each remaining index in $O(\log k)$ overall time. The data structure only requires $O\left(\frac{k \log\log k}{\log k}\right) = o(k)$ additional bits compared to $O(k \log n)$ bits required for our preceding structure. This is an important improvement for routing schemes as the RMQ structure needs to be appended to each label in order to achieve the query time improvement.

4 Color Distance Oracle supporting color-reassignments

In this section, we let $(V, \mathbf{dist})$ denote a n-point metric space and let the metric be denoted by ρ. We say that ρ is an ultrametric for V if it is a metric that ensures the strong triangle inequality $\rho(x,z) \leq \max\{\rho(x,y), \rho(y,z)\}$ for all $x, y, z \in V$. It is well-known, that a finite ultrametric can be represented by a rooted hierarchically well-separated tree (HST)[2] $T = (V_T, E_T)$ with a value assigned to each vertex in V_T by the function $\Delta : V_T \rightarrow (0, \infty)$ whose leaf set is V and where $\Delta(v) < \Delta(\textsc{Parent}(v))$ for any $v \in V_T$. Then given $x, y \in V$, $\rho(x,y) = \Delta(\textsc{lca}_T(x,y))$. Thus, working with ultrametrics is very convenient as they allow to reduce problems to problems on trees which are normally well-understood.

In 2005, Mendel and Naor [15] showed a Las Vegas algorithm that given a metric space $(V, \mathbf{dist})$ finds a ultrametric ρ such that for a subset $U \subseteq V$ of size $|U| \geq |V|^{1-1/k}$ the distortion of the ultrametric would be low, i.e. that for each $u \in U, v \in V, \mathbf{dist}(u,v) \leq \rho(u,v) \leq 128k\mathbf{dist}(u,v)$. They then showed that given this algorithm, a collection of ultrametrics $\mathcal{R} = \{\phi_1, \phi_2, .., \phi_s\}$ with $E[s] = O(kn^{1/k})$ can be found together with a function $\mathbf{home} : V \rightarrow [1, s]$ such that for each $u, v \in V$ with $i = \mathbf{home}(v)$

$$\mathbf{dist}(u,v) \leq \rho_i(u,v) \leq 128k\mathbf{dist}(u,v)$$

Recently, Abraham et al. [2] gave a deterministic algorithm that improves this result to stretch $8(1+\epsilon)k$ by increasing the size of the collection $\mathcal{R}$ by factor $O(\epsilon^{-1})$. Thus, for fixed ϵ the space only differs by a constant factor.

For our Color Distance Oracle, we find a collection of ultrametrics $\mathcal{R}$ and build for each ultrametric $\rho_i \in \mathcal{R}$ an HST T_i. Additionally, we store with each $v \in V$ a pointer to $T_{\mathbf{home}(v)}$. We observe that taking

$$\min_{u \in V_c} \Delta(\textsc{Lca}_{T_{\mathbf{home}(v)}}(u,v))$$

always gives a $8(1+\epsilon)k$ approximation on $\mathbf{dist}(v,c)$ as we take the smallest distance estimate among all estimates to colored vertices and each of them is at least $8(1+\epsilon)k$ approximate in the represented metric $\rho_{\mathbf{home}(v)}$. Let a be the least common ancestor of v and the nearest colored node in $T_{\mathbf{home}(v)}$, i.e. $a = \textsc{Lca}_{T_{\mathbf{home}(v)}}(v, \textsc{Nearest}(v,c))$. Then, there cannot be any vertex a' on the path from v to a in $T_{\mathbf{home}(v)}$ with a colored vertex in its subtree as otherwise a' would be the least common ancestor of v and the colored node and by the property of HSTs that $\Delta(x) < \Delta(\textsc{Parent}(x))$ for all $x \in V$, we would thus derive a contradiction on the minimality of $\Delta(a)$. We conclude that to derive a distance estimate on $\mathbf{dist}(v,c)$, we only need to find the nearest ancestor of v that has a colored node in its subtree. We therefore construct the data structure described in the following lemma 4 over each HST, whose proof is deferred to the full version.

[2] A good introduction to HSTs and metric representations can be found in [3].

▶ **Lemma 4.** *We can maintain a data structure over a tree $T = (V, E)$ with function $c : V \to C$ as defined before, that given $v \in V, c \in C$ finds the nearest ancestor of v in T that has a c-colored vertex in its subtree and that is able to process color-reassignments. Both operations take $O(\log\log n)$ worst-case time and the data structure requires $O(n)$ space.*

As every HST requires at most $O(n)$ space and $s = O(\epsilon^{-1}kn^{1/k})$, our data structure requires space $O(\epsilon^{-1}kn^{1+1/k})$. To achieve the data structure with fast query time, we answer a query for $\widehat{\mathbf{dist}}(v, c)$ by querying the data structure described in lemma 4 on tree $T_{\mathbf{home}(v)}$ returning the nearest colored ancestor a and we return $\Delta(a)$. Thus only a single invocation of the tree structure is required which can be implemented in time $O(\log\log n)$. For updates, we iterate over all s tree data structures and invoke the color-reassignment on the same parameters. This takes $O(\log\log n)$ time per tree and as we have s trees, the running time is bound by $O(\epsilon^{-1}\log\log nkn^{1/k})$.

For fast update times, we can process a color-reassignment $v \in V, c \in C$ by changing the color in $T_{\mathbf{home}(v)}$ only leaving v without a color in all other trees. If we run our query for $v \in V, c \in C$ on every of the s HSTs, we also know that our distance estimate is a $8(1+\epsilon)k$ approximation as we check every ultrametric and we are sure that for each vertex $u \in V, c(u) = c$ we once queried the HST $T_{\mathbf{home}(u)}$ where u is colored and $\Delta(\textsc{Lca}(v, u))$ is a $8(1+\epsilon)k$ approximation. By standard techniques [15], we can also reduce the space consumption to $O(\epsilon^{-1}n^{1+1/k})$ for the data structure with fast updates. The data structure from lemma 4 can easily be adapted to give a c-colored witness $\hat{u}$ such that $\mathbf{dist}(v, \hat{u}) \le 8(1+\epsilon)k\mathbf{dist}(v, c)$. Finally, if we are only interested in queries, we can for each vertex v of color c in a tree color all its ancestors with c where we allow multiple colors. As HSTs can be balanced, we get an additional factor of $O(\log n)$ in our data structures. We can then for each color use a succinct nearest marked ancestor structure as presented by Tsur [23] with $O(\log \frac{\log\sigma}{\log w})$ query time.

5 Lower bound for static Color Distance Oracles

In this section we prove the following lower bound.

▶ **Theorem 5.** *Consider an unweighted path $G = (V, E)$ with coloring $c : V \to C$ and $\sigma \le O(n^{1-\epsilon})$ for any $\epsilon > 0$. Then, any data structure, using space S on a machine with word size $w = \Theta(\lg n)$, reporting nearest colored node distance estimates of approximation $k = O(polylog(n))$ has query time*

$$\Omega\left(\log \frac{\log\sigma}{\log\frac{S}{n} + \log\log n}\right)$$

The theorem applies to deterministic and randomized queries, admitting a constant error probability.

Our proof extends recent results by Gawrychowski et al. [11] who proved a similar statement for the exact version of the problem. Before we prove our result, we review their lower bound and then show how to extend it.

The lower bound for the exact version of the problem is based on a reduction from the colored predecessor problem which was used to establish hardness of the predecessor problem [18, 19] in the cell-probe model. Belazzougui and Navarro showed in [4] that colored predecessor search can be reduced to the rank query problem using partial sum data structures and clever mapping. In the rank query problem, a sequence $S[1, n]$ and an alphabet $[1, \sigma]$

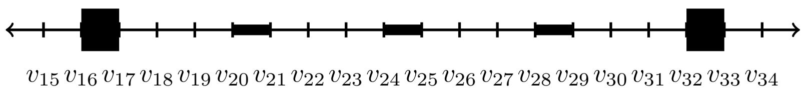

v_{15} v_{16} v_{17} v_{18} v_{19} v_{20} v_{21} v_{22} v_{23} v_{24} v_{25} v_{26} v_{27} v_{28} v_{29} v_{30} v_{31} v_{32} v_{33} v_{34}

Figure 1 The weight of the edges on our constructed path P where we chose $k = 4$. We illustrate heavy edge weight by increased boldness of the edge. For example the edge between vertices v_{16} and v_{17} has weight k^2 because $16 \mid k^2$ but $16 \nmid k^3$.

is given with each $S[i] \in [1, \sigma]$ and after the preprocessing, given an index $i \in [1, n]$ and a symbol $c \in [1, \sigma]$ the data structure has to report the number of occurrences of c in the subsequence $S[1, i)$. Finally, Gawrychowski et al. [11] observed that the rank query problem can be reduced to the nearest colored node problem as follows: Given the sequence S, we build a path $P = (v_1, .., v_n)$ where $c(v_i) = S[i]$ for every $i \in [1, n]$. With each vertex $v \in P$, we store its rank. Consider a rank query of form $i \in [1, n], c \in [1, \sigma]$. Using a Color Distance Oracle, we query for the nearest colored vertex v_j and if $j < i$, we return the rank stored at v_j. Otherwise, $j \geq i$ and we can return the rank stored with v_j decreased by 1. Let us now take the proof for theorem 5.

Proof. We first consider the (approximate) nearest colored node problem on a weighted path and extend our proof later to the unweighted setting. We assume w.l.o.g. that the approximation factor is $k \geq \log n$ and that n is an exact power of k. We use the simple construction of the path $P = (v_1, .., v_n)$ from the sequence S as before. We assume that the nearest colored vertex $v_j = \textsc{Nearest}(v_i, c)$ to v_i has $j \geq i$. This is sufficient as we can reverse the path and run a second query giving the nearest colored vertex $v_{j'}$ with $j' \leq i$ and then take the closer one (in fact one procedure might not find a colored node but we always ensure that the nearest colored node is found in one of those two queries). We let $P[v_i, v_j]$ denote the subpath of P between the vertices v_i and v_j and sometimes refer to it as interval $[i, j]$ of the path.

The intuition behind our proof is that we can choose the edge weights so that we can find the nearest colored node $v_j = \textsc{Nearest}(v_i, c)$ to v_i even if we are only given a distance estimate $\widehat{\mathbf{dist}}(v_i, c)$ of approximation factor k. Therefore, we partition the path into intervals of size k^l for every $0 \leq l \leq \log_k n$ and refer to them as level-l intervals. We assign edge weights to delimit intervals. By construction, if an edge delimits two level-l intervals then it also delimits two level-l' intervals for all $l' \leq l$. Let an edge (v_x, v_{x+1}) be assigned weight k^l if it delimits two level-l intervals but doesn't delimit two level-$l+1$ intervals. More formally, an edge (v_x, v_{x+1}) is assigned the weight k^l for the largest k^l such that $k^l \mid x$. This is depicted in figure 1. Let us now observe that the heavy edges on the path dominate the path weight. A level-l interval contains $k^{l-l'}$ level-l' intervals for $l' < l$ and therefore there are $k^{l-l'} - k^{l-l'-1}$ edges of weight $k^{l'}$. Thus the path from the first node to the last node in the level-l interval has weight

$$\sum_{l'=0}^{l-1} (k^{l-l'} - k^{l-l'-1})k^{l'} < \sum_{l'=0}^{l-1} k^{l-l'} k^{'} = \sum_{l'=0}^{l-1} k^l = lk^l \leq \log_k nk^l \leq k^{l+1}.$$

It follows that if we have $l = \lfloor \log_k(\mathbf{dist}(v_i, v_j)) \rfloor$ for $i \neq j$, then the path $P[v_i, v_j]$ contains an edge delimiting two level-$(l-1)$, i.e. of weight at least k^{l-1}, as otherwise the path costs would be strictly less than $k^l \leq \mathbf{dist}(v_i, v_j)$.

Our second idea is that given two vertices v_i, v_j with $i < j$ and path interval $[i, j]$ containing no node of color c, then the nearest colored vertex to v_j is also the nearest colored vertex to v_i, i.e. $\textsc{Nearest}(v_i, c) = \textsc{Nearest}(v_j, c)$. We want to provide some special vertices

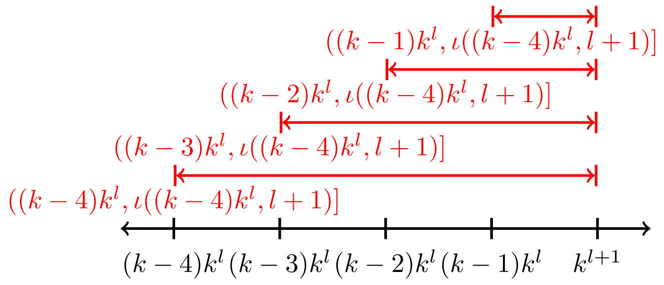

$((k-1)k^l, \iota((k-4)k^l, l+1)]$
$((k-2)k^l, \iota((k-4)k^l, l+1)]$
$((k-3)k^l, \iota((k-4)k^l, l+1)]$
$((k-4)k^l, \iota((k-4)k^l, l+1)]$
$(k-4)k^l$
$(k-3)k^l$
$(k-2)k^l$
$(k-1)k^l$
k^{l+1}

Figure 2 The drawing shows a subinterval of $[1, n]$. We see that for every number x that is divisible by k^l, we take the interval from x to the closest number that is divisible by k^{l+1} which is illustrated by the red interval.

that can return the nearest colored vertex but we need to do so carefully in order to retain near-linear space as our lower bound otherwise becomes meaningless. We therefore only cover intervals starting at special points. To simplify the presentation, we let $\iota(x, l)$ be the function that for any integer x gives the next larger integer divisible by k^l. We store with each vertex v_j where $j|k^l$ a data structure that can return $\textsc{Nearest}(v_j, c)$ for all colors c that are on the path interval $[j, \iota(j, l+1)]$. We then say that v_j covers $[j, \iota(j, l+1)]$ and observe that if j is divisible by k^l, we cover all level-l intervals starting at v_j up to the end of the current level-$(l+1)$ interval. This is also depicted in figure 2. In order to have fast look-ups, we store with each such vertex v_j a hash map with an entry for each color $c \in C$ that occurs on the path and the corresponding vertex that is closest to v_j.

It is straight-forward to see that given a vertex v_j, we can use the function ι to find $\log_k n$ special vertices $v_{\iota(j,l)}$ for $0 \le l \le \log_k n$ such that the union of all *covered* intervals by those special vertices is $\bigcup_{0 \le l \le \log_k n} [\iota(j, l), \iota(j, l+1)] = [j, n]$ because $\iota(j, l)$ is divisible by k^l by definition hence the hash map at vertex $v_{\iota(j,l)}$ covers the interval $[\iota(j, l), \iota(j, l+1)]$. Thus, we could already query the hash maps at these special vertices to extract the nearest colored node even without any distance estimate but it would incur $\log_k n$ look-ups.

Let us now combine both ideas to achieve that only a constant number of the associated hash maps at those special vertices need to be queried. Given a distance estimate $\widehat{\mathbf{dist}}(v_i, c)$ with $l = \lfloor \log_k \widehat{\mathbf{dist}}(v_i, c) \rfloor$, $v_j = \textsc{Nearest}(v_i, c)$. By our approximation guarantee

$$\mathbf{dist}(v_i, v_j) \le \widehat{\mathbf{dist}}(v_i, c) \le k\mathbf{dist}(v_i, v_j)$$

we get that $k^{l-1} \le \mathbf{dist}(v_i, v_j) \le k^{l+1}$. We conclude that it suffices to check the hash maps at the special vertices $v_{\iota(i,l')}$ from $l' \in \{l-1, l, l+1\}$ because $j \in [\iota(i, l-1), \iota(i, l+2)]$. Consider that this would not be the case, we know that the path from the interval $[i, \iota(i, l-1)]$ has path weight strictly less than k^{l-1}. If j would be in interval $(\iota(i, l+2), n]$ then as the edge $(v_{\iota(i,l+2)}, v_{\iota(i,l+2)+1})$ has weight k^{l+2} and every path to from v_i to a vertex with index in that interval has to include this edge. In both cases, we derive a contradiction.

It remains to prove that the space taken by the hash maps is near-linear. We therefore observe that the number of entries in each hash map is bounded by the size of the interval is has to cover as every node in the path interval has only one color. It is easy to see that we have $k^{\log_k n - j}$ vertices with indices divisible by k^j and each covers an interval of size k^{j+1}.

Thus the number of total entries in all hash maps can be bounded by

$$\sum_{l=0}^{\log_k n} k^{\log_k n - j} k^{j+1} = \sum_{l=0}^{\log_k n} k^{\log_k n + 1} = kn \log_k n$$

As hash maps take space linear in the number of entries and $k = O(polylog(n))$, we can bound the space by $\tilde{O}(n)$ incurring only a $\log\log n$ term in the lower bound as required. Thus, if the space is not dominated by the data structure for distance estimates, we still ensure the stated bounds.

Finally, we observe that the path $P[v_1, v_n]$ has total weight at most $\log_k n k^{\log_k n} = n \log_k n$ as it is a level-$\log_k n$ interval. We can thus replace edges of weight x with a path of $x - 1$ dummy vertices and unit weight edges. ◀

6 Lower bounds for the dynamic setting

During the last years, several techniques were presented to prove conditional lower bounds for dynamic problems by reducing to problems that are conjectured to be hard [17, 24, 1, 12]. We use the framework given by Henzinger et al. [12] who reduce their problems from a contrived version of Online-Vector-Multiplication defined as follows.

▶ **Definition 6** (γ-OuMv problem (c.f. Definition 2.6 [12])). Let $\gamma > 0$ be a fixed constant. An algorithm for the γ-OuMv problem is given parameters n_1, n_2, n_3 as its input with the promise that $n_1 = \lfloor n_2^\gamma \rfloor$. Next, it is given a matrix M of size $n_1 \times n_2$ that can be preprocessed. Let $p(n_1, n_2)$ denote the preprocessing time. After the preprocessing, a sequence of vector pairs $(u^1, v^1), .., (u^{n_3}, v^{n_3})$ is presented one vector pair after another and the task is to compute $(u^t)^\intercal M v^t$, before the pair (u^{t+1}, v^{t+1}) arrives. Let $c(n_1, n_2, n_3)$ denote the computation time over the whole sequence. The special case where $n_3 = 1$ is called the γ-uMv problem.

They then show that any algorithm solving the γ-OuMv problem in $\tilde{\tilde{o}}(n_1 n_2 n_3)$ time would give a truly subcubic algorithm to solve Online Vector Multiplication(c.f. Theorem 2.2 [12]).

As stated in the introduction, we consider in this section the existential version of nearest colored node that is we only ask whether there exists a colored vertex in the same component as a vertex v. As shown by Abboud and Williams [1] it suffices to prove lower bounds on the worst-case update and query time for a partially-dynamic version of a problem to establish amortized lower bounds for the fully-dynamic version. We thus only prove hardness of the partially-dynamic settings where we reduce from the γ-uMv problem where we are given a $n_1 \times n_2$ matrix $\mathbf{M}$ to preprocess and only a single pair of vectors $(\mathbf{u}, \mathbf{v})$ arrives.

▶ **Lemma 7.** *Given any algorithm $\mathcal{A}$ that is able to process updates in $\mathrm{U}(n, \sigma) = \tilde{\tilde{o}}(\sigma)$ and queries in $\mathrm{Q}(n, \sigma) = \tilde{\tilde{o}}(n/\sigma)$ amortized time, we can solve γ-OuMv in time $\tilde{\tilde{o}}(n_1, n_2, n_3)$. The same lower bounds extend to the worst-case update and query times of the partially-dynamic version of the problem.*

Proof. We first focus on the decremental setting and extend the proof for the fully-dynamic and incremental version. Recall that we are given a $n_1 \times n_2$ matrix $\mathbf{M}$. We treat the ith row of $\mathbf{M}$ as a subset of $[1, n_2]$, i.e. $\mathbf{M}[i, j] = 1$ iff $j \in \mathbf{M}[i]$. We create a graph G with a coloring function $c : V \to C$, with color set $C = \{c_1, .., c_{n_2}\}$, as follows: We first create a special vertex s. For every row i, we create for each $j \in \mathbf{M}[i]$ a vertex with c $c_j \in C$ and connect the vertices by linking every two consecutive vertices created. The created component for the ith row forms a simple path and is denoted from hereon by P_i. We also include an edge from s to the first vertex on the path P_i for every i. This completes the preprocessing phase. Clearly, the graph G has at most $O(n_1 n_2)$ vertices and as the construction forms a tree, we also have at most $O(n_1 n_2)$ edges. The complete set-up is depicted in figure 3.

Consider that a vector pair $(\mathbf{u}, \mathbf{v})$ arrives. For each i where $\mathbf{u}[i] = 0$, we remove the edge from the first vertex in R_i to vertex s. This incurs at most n_1 updates. We have $\mathbf{u}^\intercal \mathbf{M} \mathbf{v} = 1$ if and only if there exists a j with $\mathbf{v}[j] = 1$ where s is still connected to the color c_j. To check these connections, we need at most n_2 queries.

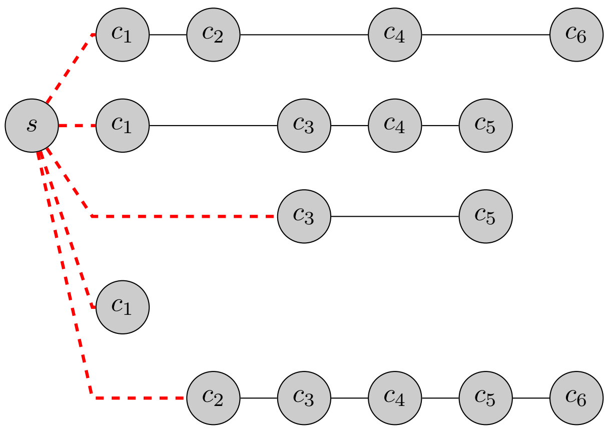


Figure 3 Depiction of the set-up of a graph from a 5×6 matrix. The point s is used as query point and in the decremental case all dashed red edges are initially in the graph and can be deleted depending on $\mathbf{u}$.

As shown by Abboud and Williams [1], we can run our algorithm on a machine that records all changes and reverts them after a single pair $(\mathbf{u}, \mathbf{v})$ is processed to the original state in time proportional to the running time since the original state was left. Our graph has at most $\sigma = n_2$ different colors. It is now straight-forward to see that we can solve γ-OuMv in time $O(n_3(n_1 \mathrm{U}(n_1\sigma, \sigma) + \sigma \mathrm{Q}(n_1\sigma, \sigma))$ thus the stated bound follows.

In the incremental setting, we omit the edges adjacent to s initially. Then, when $(\mathbf{u}, \mathbf{v})$ arrives, let $\mathbf{u}$ have the indices $u_1, u_2, ..$ set to 1. Then, we join the first vertices in R_i and R_{i+1} for even i and the last vertices of R_i and R_{i+1} for odd i. Thus, we only construct a path. Connecting s to the end of the path, allows us to run queries as in the decremental setting. For the fully-dynamic setting, we can use the same set-up as in the incremental setting but instead of rolling the edge insertions back in each phase, we can simply run edge deletions to recover the original state implying that we get an amortized bound. ◀

We underline the generality of our lower bound which establishes that even on path graphs the amortized fully-dynamic problem remains hard. To strengthen our lower bound, we observe that we can decrease the graph size to $O(n_1)$ for directed graphs or if we are given distance estimates of approximation $< 5/3$ implying that the query bound in the theorem can be replaced by $\tilde{\tilde{o}}(n)$. We therefore create $\sigma = n_2$ vertices V_c, one of each color. Instead of constructing an entire row R_i for the i'th set, we construct a single vertex v_i and connect it with edges to vertices in V_c that match elements in v_i. We can then run the algorithm as before. If we direct the edge from s to each v_i and from each v_i towards the vertices in V_c our algorithm works as before. For undirected graphs, we have that $\mathbf{dist}(v_i, c_j) = 3$ iff $\mathbf{u}^\intercal \mathbf{M} \mathbf{v} = 1$ and otherwise $\mathbf{dist}(v_i, c_j) \geq 5$. Thus any approximation of factor smaller $5/3$ is still sufficient to distinguish the two cases. Clearly the graph has $O(n_1 + n_2) = O(n_1)$ vertices as $\sigma \leq n$.

We point out that the underlying OMv-conjecture even applies in case of error probability $1/3$ thus our lower bound applies even to Monte-Carlo algorithms. Interestingly, the directed incremental version of our problem can be seen as a graph version of Pagh's problem (we follow the definition from [17]). In Pagh's problem, we are given a collection $\mathcal{C}$ of k sets $C_1, C_2, .., C_k \subseteq [n]$. We are then allowed to update by providing two indices $i, j \in \{1, .., k\}$ adding the set $C_i \cap C_j$ to $\mathcal{C}$. We then want to be able to query given $x \in [n]$, $i \in \{1, .., k\}$ if $x \in C_i$. Similarly to the proof let us assume that each set C_i is represented by a path

P_i containing a vertex of color c for each $c \in \overline{C_i}$. Then updates for $i, j \in \{1, .., k\}$ can be implemented by adding a new vertex and an edge from it to the beginning of P_i and one to the first vertex in P_j. Queries can be implemented by asking whether a node of color $x \in [n]$ can be reached from the first node of P_i and by returning the negated answer.

References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 434–443. IEEE, 2014.

2 Ittai Abraham, Shiri Chechik, Michael Elkin, Arnold Filtser, and Ofer Neiman. Ramsey spanning trees and their applications. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1650–1664. SIAM, 2018.

3 Yair Bartal, Nathan Linial, Manor Mendel, and Assaf Naor. On metric ramsey-type phenomena. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 463–472, New York, NY, USA, 2003. ACM. doi:10.1145/780542.780610.

4 Djamal Belazzougui and Gonzalo Navarro. Optimal lower and upper bounds for representing sequences. *ACM Trans. Algorithms*, 11(4):31:1–31:21, 2015. doi:10.1145/2629339.

5 Michael A. Bender and Martin Farach-Colton. The lca problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, LATIN '00, pages 88–94, London, UK, UK, 2000. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=646388.690192.

6 Shiri Chechik. Improved distance oracles and spanners for vertex-labeled graphs. In *Proceedings of the 20th Annual European Conference on Algorithms*, ESA'12, pages 325–336, Berlin, Heidelberg, 2012. Springer-Verlag. doi:10.1007/978-3-642-33090-2_29.

7 Shiri Chechik. Approximate distance oracles with constant query time. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 2014.

8 Shiri Chechik. Approximate distance oracles with improved bounds. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 1–10. ACM, 2015.

9 Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auF der Heide, Hans Rohnert, and Robert E Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM Journal on Computing*, 23(4):738–761, 1994.

10 Johannes Fischer and Volker Heun. *Theoretical and Practical Improvements on the RMQ-Problem, with Applications to LCA and LCE*, pages 36–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. doi:10.1007/11780441_5.

11 Pawel Gawrychowski, Gad M Landau, Shay Mozes, and Oren Weimann. The nearest colored node in a tree. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 54. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

12 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 21–30. ACM, 2015.

13 Danny Hermelin, Avivit Levy, Oren Weimann, and Raphael Yuster. *Distance Oracles for Vertex-Labeled Graphs*, pages 490–501. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-22012-8_39.

14 Itay Laish and Shay Mozes. Efficient approximate distance oracles for vertex-labeled planar graphs. *arXiv preprint arXiv:1707.02414*, 2017.

15 Manor Mendel and Assaf Naor. Ramsey partitions and proximity data structures. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 109–118. IEEE, 2006.

16 Shay Mozes and Eyal E Skop. Efficient vertex-label distance oracles for planar graphs. *Theory of Computing Systems*, 62(2):419–440, 2018.

17 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proc. 42nd ACM Symposium on Theory of Computing (STOC)*, pages 603–610, 2010.

18 Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 232–240. ACM, 2006.

19 Mihai Pătraşcu and Mikkel Thorup. Randomization does not help searching predecessors. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 555–564. Society for Industrial and Applied Mathematics, 2007.

20 Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *International Colloquium on Automata, Languages, and Programming*, pages 261–272. Springer, 2005.

21 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '01, pages 1–10, New York, NY, USA, 2001. ACM. doi:10.1145/378580.378581.

22 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

23 Dekel Tsur. Succinct data structures for nearest colored node in a tree. *Information Processing Letters*, 132:6–10, 2018.

24 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 645–654. IEEE, 2010.

25 Christian Wulff-Nilsen. Approximate distance oracles with improved query time. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 539–549. SIAM, 2013.

Planar Support for Non-piercing Regions and Applications

Rajiv Raman
IIIT-Delhi, Delhi, India
rajiv@iiitd.ac.in

Saurabh Ray
Department of Computer Science, NYU Abu Dhabi, United Arab Emirates
saurabh.ray@nyu.edu

Abstract

Given a hypergraph $\mathcal{H} = (X, \mathcal{S})$, a planar support for $\mathcal{H}$ is a planar graph G with vertex set X, such that for each hyperedge $S \in \mathcal{S}$, the sub-graph of G induced by the vertices in S is connected. Planar supports for hypergraphs have found several algorithmic applications, including several packing and covering problems, hypergraph coloring, and in hypergraph visualization.

The main result proved in this paper is the following: given two families of regions R and B in the plane, each of which consists of connected, non-piercing regions, the *intersection hypergraph* $\mathcal{H}_R(B) = (B, \{B_r\}_{r\in R})$, where $B_r = \{b \in B : b \cap r \neq \emptyset\}$ has a planar support. Further, such a planar support can be computed in time polynomial in $|R|, |B|$, and the number of vertices in the arrangement of the regions in $R \cup B$. Special cases of this result include the setting where either the family R, or the family B is a set of points.

Our result unifies and generalizes several previous results on planar supports, PTASs for packing and covering problems on non-piercing regions in the plane and coloring of intersection hypergraph of non-piercing regions.

2012 ACM Subject Classification Mathematics of computing → Approximation algorithms

Keywords and phrases Geometric optimization, packing and covering, non-piercing regions

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.69

Acknowledgements Part of the work was done when the first author was visiting NYU Abu Dhabi, and the author thanks the institution for its hospitality.

1 Introduction

Hypergraphs arise naturally in several applications and it is often helpful to capture the structure of the hypergraph using a *sparse graph*. One popular way to capture the structure of a hypergraph is to construct a *planar support*, which is a planar graph G with the same vertex set as the hypergraph such that every hyperedge induces a connected subgraph of G.

In this paper, we study hypergraphs that arise in several, primarily geometric settings. For example, a set of points P, and family of disks D in the plane define a hypergraph, $\mathcal{H}(P, D)$, where each disk $d \in D$ defines a hyperedge $P \cap d$. This is a widely studied hypergraph, and it is well known that the Delaunay graph of the points is a planar support for this hypergraph. Another hypergraph on the same objects is $\mathcal{H}(D, P)$, where the vertices are the disks, and each point $p \in P$ defines a hyperdge $\{d \in D : d \ni p\}$. We refer to $\mathcal{H}(P, D)$ as the *primal hypergraph*, and $\mathcal{H}(D, P)$, as the *dual hypergraph*. A hypergraph that generalizes both these hypergraphs is the following: Given a family R of *red* disks, and a family B of *blue* disks, we define the *intersection hypergraph* $\mathcal{H}(B, R)$ as follows: the disks B are the vertices, and

© Rajiv Raman and Saurabh Ray;
licensed under Creative Commons License CC-BY

each disk $r \in R$ defines a hyperedge $B_r = \{b \in B : r \cap b \neq \emptyset\}$. Both the primal and dual hypergraphs can be seen as special cases of this intersection hypergraph, by viewing the points in the arrangement as either blue disks, or red disks of radius zero. Our result implies that this intersection hypergraph has a planar support.

If instead of disks, we use topological generalizations like *pseudodisks* or *k-admissible regions*, the primal hypergraph defined above still admits a planar support [13]. However, it was not known if the same is true for the dual hypergraph. Our result implies that the intersection hypergraphs of k-admissible regions (which includes pseudodisks) admits a planar support which in particular means that the dual hypergraph has a planar support.

We go beyond k-admissible regions, and show that these results are true even if the regions have holes (i.e., they need not be simply connected[1]). Previous proofs relied heavily on the assumption that the regions are simply connected. In addition, our results are constructive and give polynomial time algorithms. Thus, even for the primal hypergraph of k-admissible regions, where a planar support was known to exist, we make progress by giving a polynomial time construction.

The existence of a planar support for the primal hypergraph for k-admissible regions has a surprising algorithmic consequence: it implies that local search yields a PTAS for the hitting set problem (the set cover problem for the dual hypergraph) [12] for k-admissible regions. Since a planar support for the dual hypergraph for k-admissible regions was not known, the authors in [1] had to construct another suitable graph in order to prove that local search yields a PTAS for the set cover problem for the primal hypergraph. Similarly, for the Dominating Set problem for k-admissible regions, an alternate graph construction was required in [1], which was a generalization of a previous construction of [7] for disks. We unify these results by considering a problem that generalizes all three problems: hitting set, set cover, and dominating set for which our result implies a PTAS. Our result is in fact stronger and is not implied by the previous results.

Prior to our result, Chan and Har-Peled [4] proved that an arrangement of k-admissible regions of depth two [2] admits a planar support. In fact, in this case, the intersection graph[3] of the regions is itself the planar support. They used this result to obtain a PTAS for the independent set problem for k-admissible regions [4]. Our result also implies PTASs for generalized versions of packing problems considered in [1] and [6]. For some of the problems, we obtain a PTAS where only a constant factor approximation was known.

In a different line of work, Keszegh [10], recently proved the following interesting result: the intersection hypergraph of two families of pseudodisks is four colorable. This result follows immediately from our result since planar graphs are four colorable. Keszegh's paper has several other results but the above result is the central tool to which most of the paper is devoted and from which the other results follow.

The result of Keszegh extends the result of Keller and Smorodinsky [9] where the hypergraph is defined by a single family of pseudodisks, the vertex set is the set of pseudodisks and the hyperedges consists of open or closed neighborhood of each pseudodisk in the intersection graph.

Before describing our results and other related results in more detail, we introduce the necessary definitions and notation.

[1] A region is simply connected if any loop can be continuously shrunk to a point while staying within the region, which is true for a disk, but not for an annulus.

[2] i.e., no more than two intersect at any point in the plane

[3] An intersection graph on a set of regions is a graph whose vertices are the regions, and two vertices are adjacent if their corresponding regions intersect.

Definitions and Notation

We will use the term *region* to refer to a set γ in the plane that can be described as $\gamma = \bar{\gamma} \setminus int(H_1 \cup \cdots \cup H_k)$, where $H_1, \cdots, H_k$ are disjoint, compact, simply connected regions contained in the compact, simply connected region $\bar{\gamma}$. Essentially, γ is a compact connected region with holes. We will refer to the region $\bar{\gamma}$ as the *filled region* corresponding to γ, and the regions $H_1, \cdots, H_k$ as the holes in γ. We will refer to the boundary of $\bar{\gamma}$ as the *outer boundary* of γ.

We say that a set of regions are in *general position* if a) the boundaries of any two of the regions intersect a finite number of times, and cross at these points, b) the boundaries of three (or more) of the regions do not intersect at a common point, and c) any region can be expanded by a small non-zero amount without changing the arrangement of the regions combinatorially (see Definition 8 in Section 2 for a formal definition of expansion of a region). Furthermore, we say that a set of points and a set of regions are together in general position if none of the points lie on the boundary of any of the regions. In the rest of the paper, we will always assume general position. Two regions γ_1 and γ_2 in the plane are said to be *non-piercing* if the sets $\gamma_1 \setminus \gamma_2$, and $\gamma_2 \setminus \gamma_1$ are both connected. A family Γ of regions is said to be non-piercing if the regions in Γ are pairwise non-piercing.

▶ Remark. The term *non-piercing* has been used previously to refer to a family of regions defined exactly as we have except that the regions are required to be simply connected (i.e., not containing holes). The term *k-admissible* refers to such non-piercing families where in addition the boundaries of each pair of regions intersect at most k times. The term *pseudodisks* is used to refer to a 2-admissible family of regions. To be more consistent with the literature, we should be using a term like "non-piercing regions with holes". However, for better readability we stick to using a shorter term.

Given a set Γ of non-piercing regions, and a set P of points in the plane, we define the *primal* hypergraph $\mathcal{H}(P, \Gamma)$ as the hypergraph in which the vertex set is P, and there is a hyperedge $\gamma \cap P$ corresponding to each $\gamma \in \Gamma$. We define the *dual* hypergraph $\mathcal{H}(\Gamma, P)$ as the hypergraph in which the vertex set is Γ, and corresponding to each $p \in P$, there is a hyperedge $\{\gamma \in \Gamma : \gamma \ni p\}$. Finally, given two families of non-piercing regions R and B, we define their *intersection hypergraph* $\mathcal{H}(B, R)$ as the hypergraph in which the vertex set is B, and corresponding to each region $r \in R$ there is a hyperedge $B_r = \{b \in B : r \cap b \neq \emptyset\}$.

1.1 Our Results and Implications

The main result we prove in this paper is the following:

▶ **Theorem 1.** *Given two families R and B of non-piercing regions, the intersection hypergraph $\mathcal{H}(B, R)$ admits a planar support.*

We now describe the implications of Theorem 1. Due to shortage of space in this extended abstract, we do not prove the claimed consequences of Theorem 1, as they follow in a straightforward manner from standard arguments.

Generalized Set Cover Problem for non-piercing regions

Given a family R of red non-piercing regions, and another family B of blue non-piercing regions, such that each $r \in R$ is intersected by at least one $b \in B$, find the smallest subset $B' \subseteq B$ such that each $r \in R$ is intersected by at least one $b \in B'$.

The following theorem below follows in a straightforward manner from the framework in [12] using Theorem 1.

▶ **Theorem 2.** *There is a PTAS for the Generalized Set Cover problem for non-piercing regions.*

When B is a set of points in the plane, this problem is equivalent to the *hitting set* problem for non-piercing regions: given a set of points and a family of non-piercing regions, find the smallest subset of points such that each region contains at least one point from our chosen subset of points. When R is a set of points in the plane, this problem is equivalent to the *set cover* problem for non-piercing regions in the plane: given a set of points and non-piercing regions in the plane, find the smallest subset of the regions so that each input point is covered by one of the chosen subset of regions. When both B and R are identical families of non-piercing regions, the problem is equivalent to the *dominating set* problem for non-piercing regions: given a family of non-piercing regions, find the smallest subset of regions so that each of the other regions intersects at least one of the chosen regions. Thus a PTAS for the generalized set cover problem for non-piercing regions implies a PTAS for all three problems: hitting set, set cover and dominating set. However, the reverse is not true: the PTASs for these three problems together do not imply a PTAS for the generalized set cover problem.

A PTAS for the hitting set problem for simply connected non-piercing regions in the plane, and halfspaces in $\mathbb{R}^3$ was given in [12]. For the set cover problem for disks in the plane a PTAS follows from the result for halfspaces, via a standard lifting to three dimensions. For the dominating set problem for disks in the plane, a PTAS was given in [7]. Generalizations of the PTASs for the set cover and dominating set problems for disks to simply connected non-piercing regions in the plane were given in [1]. None of the earlier results work in the setting where the regions are allowed to have holes.

Weighted Covering problems

In the weighted variant of the generalized set cover problem, each region $b \in B$ has a non-negative weight, and the goal is to minimize the total weight of the chosen set $B' \subseteq B$. Chan et. al. [3], building on the work of Varadarajan [14], obtained constant-factor approximation algorithms for set systems with linear *shallow-cell complexity* [4]. Note that the number of hyperedges of size 2 in $\mathcal{H}(B, R)$ is $O(|B|)$ since each such hyperedge corresponds to an edge in the planar support which cannot have more than $3|B| - 6$ edges. Since this is true for the projection of the hypergraph on any subset of B, a standard probabilistic argument due to Clarkson and Shor [5] implies that the number of hyperedges in $\mathcal{H}(B, R)$ of size at most k is $O(kn)$ implying that the shallow cell complexity of $\mathcal{H}(B, R)$ is linear. Our result thus implies a constant factor approximation for the weighted variant of the generalized set cover problem via the framework of Chan et. al. [3].

▶ **Theorem 3.** *There is an $O(1)$-approximation algorithm for the weighted Generalized Set Cover problem for non-piercing regions.*

Generalized Set Packing problem for non-piercing regions

Given a set R of red non-piercing regions, and a set B of blue non-piercing regions where each red region has a capacity bounded above by a constant $C > 0$, find a maximum cardinality subset $B' \subseteq B$, such that the number of blue regions in B' intersecting any $r \in R$ does not exceed the capacity of the region r.

[4] See [3] for the definition of Shallow Cell Complexity.

The theorem below follows in a straightforward manner from the framework in [1] using Theorem 1.

▶ **Theorem 4.** *There is a PTAS for the generalized set packing problem for non-piercing regions, when each region has a capacity bounded above by a constant.*

The generalized set packing problem specializes to the *region packing* problem when the set R is a set of points: Given a family B of non-piercing regions, and a set R of points, each with a capacity bounded above by a constant C find a maximum subset $B' \subseteq B$, such that no point $r \in R$ is covered by more than its capacity. When $C = 1$, note that the region packing problem is the discrete independent set problem which itself generalizes the independent set problem in the intersection graph of the regions in B. In [4], Chan and Har-Peled gave a PTAS for the independent set problem for a set of simply connected, non-piercing regions in the plane. When the set B is a set of points, the generalized set packing problem specializes to the *point packing problem*: Given a set of points B, and a family R of non-piercing regions, each with capacity upper bounded by a constant C, find a maximum cardinality subset of points $B' \subseteq B$, such that the number of points of B' in any $r \in R$ is at most the capacity r.

In [1], a PTAS was given for the region packing problem, when the regions were assumed to be simply connected non-piercing regions where the boundaries of each pair of regions intersect at most a constant number of times. For the point packing problem, again [1] gave a PTAS when $C = 1$. For larger values of C, only constant-factor approximation algorithms were known [6]. Our result thus extends the PTAS in [1] for any constant C. Furthermore, our PTAS for all the above problems works for non-piercing regions with holes, which generalizes the earlier results, but does not follow from them.

▶ Remark. All the PTASs mentioned above follow a local search framework, which requires the construction of a suitable graph for analysis. In earlier work, the graph construction for all the problems above relied on the fact that the regions were simply connected, and non-piercing, and did not extend to regions with holes. As far as we are aware, the above problems were not studied for non-piercing regions with holes. Further, each problem required a different graph construction. It is satisfying to finally have a unified view of all these problems.

Hypergraph Coloring

Recently, Keszegh [10], proved the following result which generalizes the result of Keller and Smorodinsky [9]: Let R and B be two families of pseudodisks. Then, the intersection hypergraph $\mathcal{H}(B, R)$ admits a coloring with 4 colors, and a *conflict-free* coloring with $O(\log n)$ colors. Keszegh's result follows from our result due to the fact that the planar support of $\mathcal{H}(B, R)$ is four colorable, and a valid coloring of the planar support is a valid coloring of the hypergraph. Our result thus extends Keszegh's result to non-piercing regions. In order to prove his result, Keszegh proves that the *Delaunay graph*, $G = (B, E)$ where the vertex set is B and E is the set of hyperedges of size 2 in $\mathcal{H}(B, R)$, is planar. Observe that our result is stronger since every edge in the Delaunay graph must be in the planar support.

▶ **Theorem 5.** *Given two families R and B of non-piercing regions, the intersection hypergraph $\mathcal{H}(B, R)$ can be colored with at most 4 colors.*

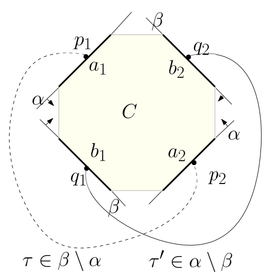


Figure 1 The assumption that there is an $\alpha, \beta, \alpha, \beta$ subsequence in the sequence of labels along ∂C leads to a contradiction.

2 Cell Bypassing

In this section, we define *cell bypassing*[5], a basic operation that is used to simplify a given arrangement of non-piercing regions. Let Γ denote a set of non-piercing regions, and let $\mathcal{A}$ denote the arrangement of these regions. We will show that whenever there is a point contained in at least 3 regions, we can simplify the arrangement by modifying a region $\gamma \in \Gamma$, while maintaining key properties required to construct a planar support.

For a region γ, we let $\partial\gamma$ denote the boundary of γ, and $int(\gamma)$ to denote the interior of γ. For a family of regions Γ, define $\partial\Gamma = \bigcup_{\gamma\in\Gamma} \partial\gamma$ where $\partial\gamma$ denotes the boundary of the region γ. The closure of each connected component of $\mathbb{R}^2 \setminus \partial\Gamma$ defines a *cell*. For a cell C, we define the following: Γ_C denotes the set of regions containing the cell C, $depth(C)$ denotes $|\Gamma_C|$, and ∂C denotes the boundary of C. If an arc of $\partial\gamma$ lies on ∂C, then the region γ is said to *contribute* to ∂C. We define *degree(C)* as the number of arcs on the boundary of C. Note that the same region may contribute multiple arcs to ∂C.

We say that two cells are adjacent if their boundaries share an arc of positive length. We define the *cell adjacency graph* [6] G_Γ of Γ as the graph in which vertices correspond to the cells and two vertices are adjacent in the graph if the corresponding cells are adjacent. Clearly, G_Γ is a planar graph. Observe that the degree of a cell in G_Γ is equal to the number of arcs on its boundary. Also note that the depth of adjacent cells C and C' in an arrangement of regions differ by exactly 1. If $depth(C) < depth(C')$, then $\Gamma_{C'} = \Gamma_C \cup \{\rho\}$, where ρ is the unique region such that the arc $C \cap C' \subseteq \partial\rho$. We say that a cell C is *maximal* if its depth is more than the depth of any cell C' adjacent to it.

▶ **Lemma 6.** *All regions in Γ contributing to the boundary of a maximal cell C in $\mathcal{A}$ contain the cell C.*

Proof. For contradiction, assume that there is an arc α of $\partial\gamma$ that appears on ∂C, but $C \not\subseteq \gamma$. Let C' be the cell adjacent to C along α. Then, $depth(C') > depth(C)$ as all regions containing C also contain C', and C' is additionally contained in γ. This contradicts the maximality of C. ◀

[5] The notion of cell bypassing is similar is spirit to the notion of *lens bypassing* in [1] and to the notion of *core decomposition* in [11]. However, the technical difference is critical for the applications in this paper.
[6] The cell adjacency graph is the geometric dual of the arrangement graph in which the intersection points of the boundaries of the regions are the vertices and two vertices are adjacent in the graph if they appear consecutively along the boundary of some region.

▶ **Lemma 7.** *Let C be a maximal simply connected cell in $\mathcal{A}$, whose boundary arcs are labelled by the regions in Γ contributing them. Let σ be the cyclic sequence of the labels of the arcs in counterclockwise order along ∂C. Then, σ is a Davenport Schinzel sequence of order 2 i.e., it does not contain a subsequence of the form $\alpha, \beta, \alpha, \beta$.*

Proof. For contradiction, let a_1, b_1, a_2, b_2 be four arcs appearing in cyclic order along ∂C, where a_1 and a_2 have the label α, and b_1 and b_2 have the label β. Since C is a maximal cell, note that $C \subseteq \alpha \cap \beta$. Let p_1 and p_2 be points in the interior of the arcs a_1 and a_2 respectively. Similarly, let q_1 and q_2 be points in the interior of the arcs b_1 and b_2 respectively. See Figure 1. Since q_1 and q_2 lie on the boundary of $\alpha \setminus \beta$ which by assumption is connected, there is a curve τ joining q_1 and q_2 whose interior lies in $\alpha \setminus \beta$. Similarly, there is a curve τ' joining p_1 and p_2 whose interior lies in $\beta \setminus \alpha$. Note that the interiors of neither τ, nor τ' intersect C, since C does not intersect either $\alpha \setminus \beta$ or $\beta \setminus \alpha$. Since p_1, q_1, p_2, q_2 appear along ∂C in that order, and since C does not have any holes (i.e., it is simply connected), τ and τ' must intersect at a point outside C and in the interior of both the curves. This is a contradiction since $\alpha \setminus \beta$ and $\beta \setminus \alpha$ are disjoint sets. ◀

▶ **Definition 8** (ϵ-expansion)**.** For any region R, define an ϵ-expansion of R, denoted R_ϵ as the Minkowski sum of R and a ball of an arbitrarily small radius ϵ centered at the origin.

▶ Remark. The ϵ-expansion of a region is necessary for technical reasons. The parameter ϵ is always chosen to be small enough so that combinatorial structure of the arrangement does not change if a region R is replaced by R_ϵ in the family Γ. Due to general position assumptions, such an ϵ always exists. The choice of ϵ thus depends on the other regions in the family Γ, but we supress this dependency for better readability.

▶ **Definition 9** (Good region)**.** For a maximal cell C in $\mathcal{A}$, a region $\gamma \in \Gamma$ is said to be a *good* region for C if the following conditions hold: i) γ contributes to the boundary of C and ii) $(\Gamma \setminus \{\gamma\}) \cup \{\gamma'\}$, where $\gamma' = \gamma \setminus int(C_\epsilon)$, is a non-piercing family.

▶ **Lemma 10.** *For any maximal simply connected cell C in $\mathcal{A}$, there is a region $\gamma \in \Gamma$ that contributes exactly one arc to the boundary of C. Furthermore, any such region is good for C.*

Proof. First we argue that there is a region that contributes exactly one arc to ∂C. If every arc on ∂C has a distinct label, we are done. Otherwise, consider two arcs a_1 and a_2 having the same label α that are closest to each other in counterclockwise order along ∂C. Let b be any arc lying between a_1 and a_2. There is such an arc since consecutive arcs along ∂C cannot have the same label. Let γ be the label of the arc b. By the choice of a_1 and a_2 and Lemma 7 there cannot be any other arc on ∂C with label γ. Thus, there is at least one region γ that contributes exactly one arc to ∂C.

We now show that any such region γ is a good region for C. The fact that γ contributes exactly one arc to ∂C, along with the fact that C is simply connected implies that $\gamma' = \gamma \setminus int(C_\epsilon)$ is connected. For any other region $\nu \in \Gamma$, we now argue that both $\nu \setminus \gamma'$ and $\gamma' \setminus \nu$ are connected. Suppose first that ν does not contain C. Then $\nu \setminus \gamma' = \nu \setminus \gamma$ which is connected. Also $\gamma' \setminus \nu = (\gamma \setminus \nu) \setminus int(C_\epsilon)$, which is connected since the boundaries of $\gamma \setminus \nu$ and C intersect only on one arc along ∂C. Now suppose that ν contains C. Then note that $\gamma' \setminus \nu$ is almost the same as $\gamma \setminus \nu$. In fact if ν does not contribute to the boundary of C, then $\gamma' \setminus \nu = \gamma \setminus \nu$. Otherwise, $\gamma' \setminus \nu$ is obtained by shaving off a thin strips of width ϵ from the boundary of $\gamma \setminus \nu$. Thus $\gamma' \setminus \nu$ is connected. Also $\nu \setminus \gamma' = (\nu \setminus \gamma) \cup C_\epsilon$. Since $\nu \setminus \gamma$ and C_ϵ are both connected, and have a non-empty intersection, their union is also connected. ◀

▶ **Lemma 11.** *Any maximal cell C in $\mathcal{A}$ of depth at least two, and containing a hole, has exactly one hole H. Exactly one region $\gamma \in \Gamma$ contributes to the boundary of H* [7]*. This region γ does not contribute any other arc to the boundary of C, and is a good region for C.*

Proof. First, observe that the boundary of any hole in C can be contributed to by at most one region. To see this assume to the contrary that two or more regions contribute to the boundary of a hole in C. If the boundaries of two of these regions intersect on the boundary of the hole, then the boundaries of those regions intersect the interior of the cell, which is impossible. Otherwise the hole belongs to at least two distinct regions, which violates the general position assumption.

Let H_γ be a hole in C whose boundary is contributed to by the region γ. We will show that H_γ is the only hole in C. If γ contributed to another hole, say H'_γ in C, then for any other region β (such a region exists, since $depth(C) \geq 2$) containing C intersects $int(H_\gamma)$, as well as $int(H'_\gamma)$, and thus $\beta \setminus \gamma$ cannot be connected. Thus γ cannot contribute to any other hole in C.

Let ρ be any other region containing C. Then, note that ρ intersects $int(H_\gamma)$ and since C is contained in both γ and ρ, we must have $\rho \setminus \gamma \subset int(H_\gamma)$, as otherwise $\rho \setminus \gamma$ would not be connected. This implies that $\rho \subset int(\gamma \cup H_\gamma)$. In particular, this means that $\rho \subset int(\bar{\gamma})$, where $\bar{\gamma}$ is the outer boundary of the region γ. If ρ contributed to the boundary of another hole H_ρ in C, then by the same argument, we would have $\gamma \subset int(\bar{\rho})$, a contradiction. This means that H_γ is the only hole in C.

Since the depth of C is at least two, there is at least one other region ρ containing C. However, as argued before $\rho \subset int(\gamma \cup H_\gamma)$. Since $C \subseteq \rho$ the boundary of γ cannot contribute any other arc (other than the boundary of H_γ) to the boundary of C.

We now argue that γ is a good region for C. Let $\gamma' = \gamma \setminus C_\epsilon$. The region γ' is connected since γ' is obtained from γ by replacing the hole H_γ by a larger hole $H = C_\epsilon \cup H_\gamma \subset int(\bar{\gamma})$ [8] containing H_γ. H is simply connected since H_γ is the only hole in C. Also note that no other hole of γ intersects H. Let ν be any other region in Γ. We will show that both $\nu \setminus \gamma'$, and $\gamma' \setminus \nu$ are connected. Suppose first that $\nu \cap C = \emptyset$. Then, $\nu \setminus \gamma' = \nu \setminus \gamma$, which by assumption, is connected. Also, note that $\gamma' \setminus \nu$ is obtained by replacing the hole H_γ in $\gamma \setminus \nu$ by the larger hole H which contains H_γ, is contained in $int(\bar{\gamma})$, and does not intersect any other holes in $\gamma \setminus \nu$. Thus $\gamma' \setminus \nu$ is connected. Now suppose that $C \subset \nu$. Then $\gamma' \setminus \nu$ is almost the same as $\gamma \setminus \nu$ and is obtained by shaving off a thin strip of width ϵ from the boundary of $\gamma \setminus \nu$. Thus $\gamma' \setminus \nu$ is connected. Also, $\nu \setminus \gamma' = (\nu \setminus \gamma) \cup C_\epsilon$. Since $\nu \setminus \gamma$ intersects C_ϵ near the boundary of the hole H_γ, $\nu \setminus \gamma'$ is connected. ◀

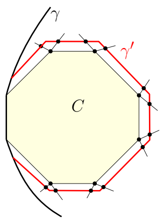
Lemmas 10 and 11 imply the following lemma.

▶ **Lemma 12.** *Any maximal cell C of depth at least two in the arrangement $\mathcal{A}$ has a good region.*

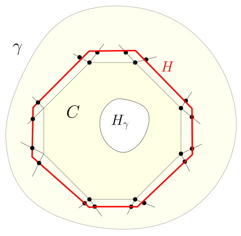
▶ **Definition 13** (Cell Bypassing)**.** Let C be a maximal cell and γ be a good region for C. Then, by cell-bypassing of C by γ, we mean the modification of γ to $\gamma' = \gamma \setminus int(C_\epsilon)$. See Figures 2a and 2b.

The following observations are immediate.

[7] In other words H is a hole of γ.
[8] Recall that γ does not contribute to the outer boundary of C.



(a) Case 1: C is simply connected. The portion of the boundary of γ' distinct from γ is shown in red.



(b) Case 2: C is not simply connected. The hole H_γ of γ is expanded to H, so that it now contains C.

Figure 2 Bypassing of a cell C by a good region γ.

▶ **Observation 14.** *If a maximal cell C is identical to a region γ, then γ is good for C and cell-bypassing replaces γ by an empty region, effectively removing γ from the family of regions.*

▶ **Observation 15.** *When a cell C of depth d is bypassed by a region γ, the number of cells of depth d decreases by 1. The number of cells of lower depth may increase. In fact, there are at most $O(\Delta)$ newly created cells of depth either $d-2$ or $d-3$, where Δ is the degree of the cell C.*

3 Construction of a planar support

We first give the construction of a planar support for the dual hypergraph defined by a set of non-piercing regions, and all points in the plane. We then use this to construct a planar support for the intersection hypergraph of two non-piercing families.

3.1 Planar Support for Dual Hypergraph

In the following, let Γ be a family of non-piercing regions and let $\mathcal{A}$ denote their arrangement. We construct a planar support for the hypergraph $\mathcal{H}(\Gamma, \mathbb{R}^2)$ [9], which immediately implies a planar support for any $P \subseteq \mathbb{R}^2$. For a set of $S \subseteq \Gamma$ of the regions and any point $p \in \mathbb{R}^2$, we denote the set of regions in S that contain p by S_p. Similarly for a cell C, S_C denotes the set of regions in S containing C. For a graph G and a subset U of the vertices of G, we denote the subgraph of G induced by U by $G[U]$.

▶ **Lemma 16.** *Let C be a maximal cell in $\mathcal{A}$ with $depth(C) \geq 3$. Let Γ' be the arrangement obtained by cell bypassing of C by a good region γ of C. Then, a planar support for $\mathcal{H}(\Gamma, \mathbb{R}^2)$ can be constructed from a planar support for $\mathcal{H}(\Gamma', \mathbb{R}^2)$.*

Proof. Suppose ∂C consists of a single arc. Then, C is identical to some region $\gamma \in \Gamma$. Since $depth(C) \geq 3$, γ is completely contained in another region $\rho \in \Gamma$. Bypassing C in this case results in the arrangement $\Gamma' = \Gamma \setminus \{\gamma\}$. Given a planar support G' for $\mathcal{H}(\Gamma', \mathbb{R}^2)$, we can obtain a planar support G for $\mathcal{H}(\Gamma, \mathbb{R}^2)$ by simply adding a new vertex for γ and an edge between γ and ρ. Since we obtain G by adding a vertex of degree 1 to the planar graph G', G is planar. To see that G is a support, consider any point p in the plane. If $p \notin \gamma$ then $\Gamma_p = \Gamma'_p$ and therefore $G[\Gamma_p] = G'[\Gamma'_p]$ which is connected. On the other hand, if $p \in \gamma$,

[9] While there are infinitely many points, the hypergraph is still finite, since all points in the same cell of the arrangement $\mathcal{A}$ define the same hyperedge.

then $\Gamma_p = \Gamma'_p \cup \{\gamma\}$. Since G' is a planar support for Γ', $G[\Gamma'_p] = G'[\Gamma'_p]$ is connected. Since $\rho \in \Gamma'(p)$ and there is an edge between γ and ρ in G, it follows that $G[\Gamma(p)]$ is connected.

Now, suppose ∂C contains at least two arcs. Let $\rho \neq \gamma$ be a region that contributes to ∂C. Then, in the arrangement of Γ, there is a cell X adjacent to C such that $\Gamma_X = \Gamma_C \setminus \{\rho\}$. After γ is modified to bypass C, in the arrangement of Γ', there is still a cell X' such that $\Gamma'_{X'} = \Gamma_X$ and there is a cell C' s.t. $\Gamma'_{C'} = \Gamma_C \setminus \{\gamma\}$. Since $|\Gamma_C| \geq 3$, the sets $\Gamma'_{X'}$ and $\Gamma'_{C'}$ intersect. Also note that their union is Γ_C. Since $G'[\Gamma'_{X'}]$ and $G'[\Gamma'_{C'}]$ are connected, it follows that $G'[\Gamma_C]$ is connected. Thus $G = G'$ is a planar support for $\mathcal{H}(\Gamma, \mathbb{R}^2)$. ◀

The following lemma follows from [4]. The intersection graph of a set of regions is a graph in which the vertices correspond to the regions and edges correspond to pairs of intersecting regions.

▶ **Lemma 17** ([4]). *If Γ is a family of non-piercing regions so that no cell in their arrangement has depth more than 2, then the intersection graph of Γ is planar, and is a support for $\mathcal{H}(\Gamma, \mathbb{R}^2)$.*

▶ Remark. Even though the proof presented in [4] is for k-admissible regions, essentially the same proof works for non-piercing regions (with holes).

▶ **Theorem 18.** *Let Γ be a family of non-piercing regions. Then, there exists a planar support for $\mathcal{H}(\Gamma, \mathbb{R}^2)$.*

Proof. For any set of regions Γ, define d_Γ to be the maximum depth of any cell in the arrangement of Γ, and n_Γ to be the number of cells in the arrangment with depth d_Γ. We use induction on the pair (d_Γ, n_Γ). If $d_\Gamma \leq 2$, then by Lemma 17, the intersection graph of Γ is the required planar support. Given a set of regions with $d_\Gamma \geq 3$, let Γ' be the set of regions obtained after bypassing a cell of maximum depth in the arrangement of Γ. Then, by Observation 15, the pair $(d_{\Gamma'}, n_{\Gamma'})$ is lexicographically smaller than (d_Γ, n_Γ). Inductively, a planar support for $\mathcal{H}(\Gamma', \mathbb{R}^2)$ exists, which by Lemma 16, is also a planar support for $\mathcal{H}(\Gamma, \mathbb{R}^2)$. ◀

3.2 Planar Support for the Intersection Hypergraph

Given a family R of *red* non-piercing regions, and a family B of *blue* non-piercing regions, we prove that there exists a planar support for the intersection hypergraph $\mathcal{H}(B, R) = (B, E)$, where $E = \{B_r : r \in R\}$, and $B_r = \{b \in B : b \cap r \neq \emptyset\}$. However, it is not essential for the rest of the paper. Note that we can assume without loss of generality that any red region intersects at least one blue region, and we make this assumption throughout this section. This implies that any maximal cell in the arrangement of $R \cup B$ has depth at least 2.

Let $B_{|r} = \{b \cap r : b \in B_r\}$ be the set of regions obtained by intersecting the regions in B with the region r. A region in $B_{|r}$ may have multiple connected components, but we still treat it as a single region. Let $G_{|r}(B)$ be the intersection graph of these regions. We start with the following simple observation.

▶ **Lemma 19.** *If for each red region $r \in R$, the graph $G_{|r}(B)$ is connected, then the planar support for the hypergraph $\mathcal{H}(B, \mathbb{R}^2)$, is also a planar support for the hypergraph $\mathcal{H}(B, R)$.*

Proof. Let G be the planar support of the hypergraph $\mathcal{H}(B, \mathbb{R}^2)$ defined by the blue regions. Recall that the graph G guarantees that for each $p \in \mathbb{R}^2$, the set of blue regions containing p induce a connected subgraph of G. Consider any red region $r \in R$. Since, by assumption, the graph $G_{|r}(B)$ is connected, for any pair of blue regions $s, t \in B_r$ intersecting r, there is

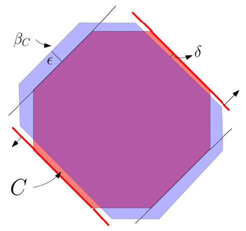


Figure 3 The construction of β_C, for a non-maximal cell C not contained in a blue region in B.

a sequence $s = b_1, \cdots, b_k = t$, of regions in B_r such that adjacent regions b_i, b_{i+1} intersect at a point $q_i \in r$. This means that there is a path in G between b_i and b_{i+1} via the regions containing q_i, i.e., via regions in B_r. This in turn implies that there is a path between s and t in G via regions in B_r. ◀

The construction of the planar support in the general setting is a reduction to the setting in Lemma 19.

▶ **Lemma 20.** *Given a set of red and blue non-piercing regions R and B, we can obtain a modified set of red regions R', such that* (i) *the set of regions R' is non-piercing,* (ii) *in the arrangement of $R' \cup B$, each maximal cell is contained in some blue region in B, and* (iii) *the intersection hypergraph $\mathcal{H}(B, R')$ is isomorphic to $\mathcal{H}(B, R)$.*

Proof. Consider a maximal cell C in the arrangement of $R \cup B$ that is not contained in any blue region. Then, C is also a maximal cell in the arrangement of R. Since R is a non-piercing family, by Lemma 12 we can bypass C. This does not change the intersection hypergraph of the red and blue regions, since no red-blue intersection is lost or gained as a consequence of bypassing the cell C. We repeat this process until each maximal cell is in some blue region. The modified set of regions thus obtained satisfy the conditions of the lemma. ◀

▶ **Lemma 21.** *Given two families of red and blue non-piercing regions R and B, such that each maximal cell in the arrangement $\mathcal{A}$ of $R \cup B$ is contained in some region $b \in B$, we can add a* fake *blue region β_C corresponding to each cell C in $\mathcal{A}$ that is not contained in any blue region in B, such that R, B' and B'', where B'' is the set of fake blue regions and $B' = B \cup B''$, satisfy:* (i) *B' is a family of non-piercing regions,* (ii) *each β_C intersects only those regions in R that contain C, and* (iii) *for each $r \in R$, the intersection graph $G_{|r}(B')$ is connected.*

Proof. Let C be a cell not contained in any of the blue regions. We define the fake blue region for this cell as $\beta_C = C_\epsilon \setminus \bigcup_{r \in R,\, C \not\subset r} int(r_\delta)$, where $\delta < \epsilon$ and ϵ is sufficiently small. See Figure 3. Intuitively, β_C is roughly the same as C but we modify its boundary slightly so that it intersects all the blue regions that contribute to the boundary C but does not intersect any red regions not containing C. Defining β_C this way ensures that property (ii) in the statement of the Lemma is satisfied. Choosing ϵ to be sufficiently small also ensures that property (i) is satisfied. Finally, choosing $\delta < \epsilon$ also ensures that each red region in R is covered by the union of the blue regions in B' which implies property (iii). To see this, consider a point p contained in $r \in R$, and let D be the cell in $\mathcal{A}$ containing p. If p is not contained in a blue region in B, then since D does not lie in any blue region, we add a fake blue region β_D corresponding to D. If p lies at a distance of at least δ from any arc of ∂D, contributed by a red region in R not containing D, then note that p lies in β_D. Otherwise, if p lies within distance δ of some arc α contributed by a red region not containing D, the cell D' adjacent to D sharing the arc α is also not contained in any blue region in B. In this case, since $\epsilon > \delta$, $\beta_{D'}$ contains p. This implies that $G_{|r}(B)$ is connected for each $r \in R$. ◀

We are now ready to prove Theorem 1.

Proof of Theorem 1. If $G_{|r}(B)$ is connected for each $r \in R$, then we obtain a planar support by Lemma 19. If not, let R' be the set of modified red regions obtained by applying Lemma 20. Let B'' be the set of fake blue regions added by applying Lemma 21 to the blue regions B, and the modified red regions R'. Now, by Lemma 19, a planar support for $\mathcal{H}(B', \mathbb{R}^2)$ is a planar support for $\mathcal{H}(B', R')$. Let G' be this planar support. We show that we can obtain a planar support G for $\mathcal{H}(B, R')$, by suitably modifying G'. G is also a support for $\mathcal{H}(B, R)$ since $\mathcal{H}(B, R)$ is isomorphic to $\mathcal{H}(B, R')$. In the following, we refer to a vertex in the support graph by the corresponding region. Let $\mathcal{A}$ be the arrangement of $B \cup R'$, and let $\mathcal{B}'$ denote the arrangement of the regions in B'. Let C be a cell in $\mathcal{A}$ not contained in any blue region in B. By Lemma 20, C is not a maximal cell. Thus, there is a fake blue region $\beta_C \in B'$ corresponding to C. Since C is not maximal, we can pick a cell C' (arbitrarily chosen in case of ties) adjacent to C in $\mathcal{A}$, such that $depth(C') = depth(C) + 1$. Let b be a fake or real blue region defined as follows: if C' is not contained in any blue region in B then $b = \beta_{C'}$, otherwise b is the unique blue region in B containing C'.

Note that b forms a depth 2 intersection with β_C in $\mathcal{B}'$ (i.e. there is a point in the plane contained in just these two regions), and therefore, β_C and b are adjacent in G'. We orient the corresponding edge in G' from β_C to b. Thus, for each fake region in G', there is exactly one outgoing oriented edge incident on it. Note that not all edges in G' are oriented.

By Property (ii) of Lemma 21, any red region intersecting β_C contains C. Such a region also contains C' since C and C' are adjacent cells in $\mathcal{A}$, and $depth(C') = depth(C) + 1$. Thus all red regions intersecting β_C also intersect b. In other words, the set of red regions in R' intersecting β_C is a subset of the red regions in R' intersecting b. This is a key property we will use later. By Lemma 20, each maximal cell in $\mathcal{A}$ is contained in a blue region in B, and therefore there is a unique directed path starting from a fake region β_C and ending at a real blue region $\tilde{b} \in B$. Crucially, the set of red regions intersecting any fake region β_C is a subset of the red regions intersecting $\tilde{b}$ due to the key property mentioned earlier.

The set of oriented edges in G' form a spanning forest, where each arc is oriented towards the root of a tree, and the root of each tree corresponds to a real region. We obtain G by contracting the edges in the forest (i.e., all oriented edges in G), effectively merging all nodes in a sub-tree to its root. Since edge contraction preserves planarity, it follows that G is planar. To see that G is a support for $\mathcal{H}(B, R')$, let $b, b' \in B$ be a pair of blue regions intersecting a red region $r \in R'$. Since G' is a planar support for $\mathcal{H}(B', R')$, there is a path $b = b_1, \ldots, b_k = b'$ such that each region b_i in the path intersects r. Since each fake region on the path is merged with a real region that also intersects r, it follows that there is a path in G between b and b', such that all regions along the path intersect r.

Finally, by Lemma 20, since $\mathcal{H}(B, R')$ is isomorphic to $\mathcal{H}(B, R)$, G is the desired planar support. ◀

3.3 Algorithms

The proofs for the existence of the planar supports given in Section 3 are constructive and can be converted to an algorithm with running time $O(n^3 + m)$ where n is the number of regions and m is the number of vertices in the arrangement of the regions. Note that if the boundaries of every pair of regions intersect at most $O(n)$ times, then m is $O(n^3)$. While the applications mentioned in this paper require only the existence of a planar support, the algorithmic question is natural and may have applications in hypergraph visualization (see [2] and [8]). In the proofs of the existence of the planar support, we modify the arrangement of the regions by doing cell bypassing. For algorithmic purposes, instead of maintaining the geometric shapes of the regions, we can maintain the dual arrangement graph. In addition, we orient the edges from a cell of lower depth to a cell of higher depth and label the edge by the index of the region whose boundary separates the cells. The vertex corresponding to a cell also stores the depth of the cell and pointers to neighboring vertices. The size of the graph is $O(m)$. We assume that the initial dual arrangement graph H of the input set of region Γ is given. It is not at all apparent that these algorithms run in polynomial time, since the number of cells in the arrangement can increase after each cell-bypassing. However, using a more detailed combinatorial analysis, we can define a suitable potential function on the arrangement of the regions, so that each time we do a cell bypassing operation the potential goes down by an amount proportional to the time spent in the cell-bypassing operation. The initial potential can be shown to be $O(n^3 + m)$ which implies the upper bound on the running time. The details of the analysis are not included due to lack of space.

References

1 Aniket Basu Roy, Sathish Govindarajan, Rajiv Raman, and Saurabh Ray. Packing and covering with non-piercing regions. *Discrete & Computational Geometry*, Mar 2018. doi:10.1007/s00454-018-9983-2.

2 Kevin Buchin, Marc Van Kreveld, Henk Meijer, Bettina Speckmann, and Kevin Verbeek. On planar supports for hypergraphs. In *International Symposium on Graph Drawing*, pages 345–356. Springer, 2009.

3 Timothy M. Chan, Elyot Grant, Jochen Könemann, and Malcolm Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1576–1585, 2012.

4 Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012.

5 Kenneth L Clarkson and Peter W Shor. Applications of random sampling in computational geometry, ii. *Discrete & Computational Geometry*, 4(5):387–421, 1989.

6 Alina Ene, Sariel Har-Peled, and Benjamin Raichel. Geometric packing under non-uniform constraints. In *Proceedings of the Twenty-eighth Annual Symposium on Computational Geometry*, SoCG '12, pages 11–20, New York, NY, USA, 2012. ACM.

7 Matt Gibson and Imran A. Pirwani. Algorithms for dominating set in disk graphs: Breaking the $\log n$ barrier. In *Algorithms – ESA 2010 - 18th Annual European Symposium, Liverpool, United Kingdom, September 6–8, 2010, Proceedings*, pages 243–254, 2010.

8 David S Johnson and Henry O Pollak. Hypergraph planarity and the complexity of drawing venn diagrams. *Journal of graph theory*, 11(3):309–325, 1987.

9 Chaya Keller and Shakhar Smorodinsky. Conflict-free coloring of intersection graphs of geometric objects. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2397–2411, 2018. doi:10.1137/1.9781611975031.154.

10 Balázs Keszegh. Coloring intersection hypergraphs of pseudo-disks. In *Proceedings of the Thirty-fourth International Symposium on Computational Geometry*, 2018. URL: http://arxiv.org/abs/1711.05473.

11 Nabil H. Mustafa, Rajiv Raman, and Saurabh Ray. Quasi-polynomial time approximation scheme for weighted geometric set cover on pseudodisks and halfspaces. *SIAM Journal on Computing*, 44(6):1650–1669, 2015.

12 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.

13 Evangelia Pyrga and Saurabh Ray. New existence proofs for ϵ-nets. In *Proceedings of the Twenty-fourth Annual Symposium on Computational Geometry*, SoCG '08, pages 199–207, New York, NY, USA, 2008. ACM.

14 Kasturi Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pages 641–648, 2010.

An Exact Algorithm for the Steiner Forest Problem

Daniel R. Schmidt[1]
Institut für Informatik, Universität zu Köln, Germany
schmidt@informatik.uni-koeln.de
https://orcid.org/0000-0001-7381-912X

Bernd Zey
Fakultät für Informatik, TU Dortmund, Germany
bernd.zey@tu-dortmund.de

François Margot
Carnegie-Mellon-University, Pittsburgh PA, USA

Abstract

The Steiner forest problem asks for a minimum weight forest that spans a given number of terminal sets. The problem has famous linear programming based 2-approximations [1, 15, 20] whose bottleneck is the fact that the most natural formulation of the problem as an integer linear program (ILP) has an integrality gap of 2. We propose new cut-based ILP formulations for the problem along with exact branch-and-bound based algorithms. While our new formulations cannot improve the integrality gap, we can prove that one of them yields stronger linear programming bounds than the two previous strongest formulations: The directed cut formulation [2, 7] and the advanced flow-based formulation by Magnanti and Raghavan [25]. In an experimental evaluation, we show that the linear programming bounds of the new formulations are indeed strong on practical instances and that our new branch-and-bound algorithms outperform branch-and-bound algorithms based on the previous formulations. Our formulations can be seen as a cut-based analogon to [25], whose existence was an open problem.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization

Keywords and phrases branch-and-bound algorithms, Steiner network problems

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.70

Related Version A preliminary version is available at `https://arxiv.org/abs/1709.01124`.

1 Introduction

The Steiner forest problem (SFP) is one of the fundamental network design problems. Given an edge-weighted undirected graph $G = (V, E)$ and terminal sets $T^1, \ldots, T^K \subseteq V$, it asks for a minimum weight forest in G such that the nodes inside each terminal set are connected. Steiner forest is a particularly important problem in the design of real-world communication networks where unwieldy additional constraints make it hard to obtain hard guarantees and clean approximation algorithms. Instead, linear programming based branch-and-bound (B&B) algorithms are a popular choice here: They find an *optimum* solution in (worst-case) exponential time, but can also be run in a heuristic mode where the algorithm stops with a sub-optimum solution after a given time limit. In the latter case, B&B algorithms still

[1] Supported by a fellowship in the Postdoc-Program of the German Academic Exchange Service (DAAD).

© Daniel R. Schmidt, Bernd Zey, and François Margot;
licensed under Creative Commons License CC-BY

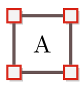
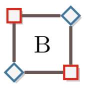
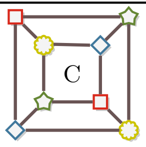
formulation			
undirected flow/cut ($\mathfrak{L}^{uc}$), lifted cut ($\mathfrak{L}^{klsvz}$) [23]	2	2	4
layered directed, $\mathfrak{L}^{dc}$ [2, 7]	3	2	4
Magnanti-Raghavan [25], $\mathfrak{L}^{mr}$	3	3	6
our extended cut-based, $\mathfrak{L}^{edc}$	3	2.5	5.14
our strengthened extended cut-based, $\mathfrak{L}^{sedc}$	3	3	6
integer optimum	*3*	*3*	*7*

Figure 1 A comparison of lower bounds from LP relaxations. The terminal sets of the three Steiner forest instances are depicted in different shapes (□, ◇, ☆, and ○). All edges have unit cost.

provide a *per-instance* quality guarantee by linear programming. This per-instance guarantee is appealing to practitioners: While often only a few selected instances need to be solved, it is common that they defy theoretical analysis. Hence, B&B algorithms complement the fixed parameter tractable (FPT) algorithms which exploit special structures of practical instances. In contrast, however, B&B algorithms allow us to include the real-world constraints without additional analyses and make no structural assumptions. In this way, they provide another important tool for problem solving in practice.

A linear programming based B&B algorithm systematically finds an optimum solution to an integer linear program (ILP). Assume that we are minimizing. The algorithm first removes the integrality requirement, turning the NP-hard ILP into a polynomial time solvable linear programming (LP) relaxation. Any solution to the ILP is a solution to the LP relaxation and thus, the value of any LP solution x^* is a lower bound on the optimum value of the ILP. If x^* is integral, we found an optimum solution to the ILP. Otherwise, there is at least one fractional variable, say $x_i^* \notin \mathbb{Z}$. In any optimum integral solution, we have either $x_i \leq \lfloor x_i^* \rfloor$ or $x_i \geq \lceil x_i^* \rceil$. We create a subproblem for each of the two cases and recurse the algorithm. If at any point in the recursion (the *branch-and-bound tree*) we obtain an integral solution or if a subproblem turns out to be infeasible, we solved the subproblem and we can prune the corresponding branch from the tree. We keep track of the best integral solution we find (the *incumbent*), as it provides an upper bound on the value of an optimum ILP solution. Since the LP value of each subproblem provides a lower bound for its optimum integral value, we can equally prune the tree once the LP value rises above the value of the incumbent. This highlights why strong LP relaxations are paramount for B&B algorithms: The better the LP bounds, and the sooner the LP relaxation becomes integral, the sooner the recursion can be pruned. As different ILP formulations for the same problem yield different LP bounds and finding a strong ILP formulation is an interesting challenge.

While B&B algorithms for the Steiner forest problem all follow the same basic algorithm, they differ on the LP relaxation they employ to generate lower bounds. From a theoretical perspective, almost all LP relaxations for the Steiner forest problem have a worst-case integrality gap of 2, with the only known exception being the *lifted cut relaxation* by Könemann, Leonardi, Schäfer, and van Zwam [23] that achieves a gap of $2 - \varepsilon$. Still, the different LP relaxations do not all yield equally good bounds: The bound from the *directed cut* relaxation will never be worse (and often much better) than the bound from the *undirected cut* relaxation, since the former is a specialization of the latter. Likewise, Magnanti and Raghavan [25] show that their *improved flow* relaxation is always as least as good as the undirected cut relaxation. In that sense, some relaxations are stronger than others, while others are incomparable (see Figure 1): The lifted cut relaxation is at least as strong as the undirected cut formulation, but it may yield stronger *or* weaker bounds than the directed variant. Experiments support this notion of relaxation strength. For instance, it has been observed that the *directed cut formulation* is better suited for B&B algorithms than the *undirected cut formulation* [6], at least for the Steiner tree problem (the special case where $K = 1$). Magnanti and Raghavan obtain particularly strong bounds from the improved flow formulations in their experiments where their B&B algorithm can solve many instances without having to branch. The bounds from the lifted cut relaxation are identical to the ones from the undirected cut relaxation in our experiments.

Our contribution. The above observations seem to turn the improved flow formulation and the directed cut formulation into the canonical choices for a B&B algorithm. Unfortunately, the improved flow relaxation is exponentially large and it is unknown if it can be solved efficiently. The directed cut relaxation is easy to solve, but its bounds are considerably weaker if used for the Steiner forest problem (an analysis is given in Section 2).

We propose a branch-and-bound algorithm that is based on a new, cut-based ILP formulation for the Steiner forest problem. Its LP relaxation is stronger than the improved flow relaxation and as the directed cut relaxation, and therefore, as the undirected cut relaxation as well. In contrast to the improved flow formulation it can be solved in polynomial time. This answers an open problem in [25] which asks for a cut-based ILP formulation that is at least as strong as the improved flow formulation. In our experiments the LP bounds are stronger than what can be achieved from any of the previous relaxations. They can also be computed quickly and reliably. Using known techniques and a computational analysis, we engineer our branch-and-bound algorithm to solve all medium sized and almost all large instances from the benchmark set. The algorithm outperforms B&B algorithms based on the previous formulations. Figure 1 shows a comparison of the formulations on widely-used small example instances.

While we focus on B&B algorithms here, new integer linear programming formulations are interesting beyond exact algorithms: Current approximation algorithms for the Steiner forest problem with a guarantee of 2 are based on iterative rounding [20] and the primal-dual analysis technique [1, 15]; two techniques that rely on strong LP relaxations. Even though our new formulation has an integrality gap of 2 as well and thus cannot directly improve the known LP-based approximation algorithms, we believe that it can inspire new research in this direction.

Related work. The directed cut relaxation for the Steiner tree problem [2, 6, 22] can be trivially extended to the Steiner forest case. It cuts off fractional solutions by imposing a direction on each edge, looking for a rooted directed tree that connects all terminals. In the Steiner *tree* case where only one terminal set exists, this process is straight-forward. When multiple sets are present, however, one directed tree per set is needed and these, in general, can impose conflicting orientations to the edges. This is a major additional difficulty in solving the Steiner forest problem. Magnanti and Raghavan [25] show how to consolidate the conflicts with the improved flow formulation.

The issues with conflicting orientations can be avoided altogether by using strong undirected formulations. Goemans [13], Lucena [24], as well as Margot, Prodon and Liebling [26] independently propose an ILP formulation for the Steiner *tree* problem that builds on Edmond's complete description of the tree polytope [11]. This tree-based formulation has a straight-forward extension to the Steiner *forest* problem, but its LP bounds are identical to the ones from the directed formulations.

The literature for the Steiner *tree* problem is more extensive: Several surveys compare ILP formulations and their polyhedral properties [7, 8, 14, 27, 28]. They are the basis for B&B algorithms [6, 22]. Exact FPT algorithms identify parameters that make the problem difficult to solve [4, 9, 12, 19]. Similarly, preprocessing techniques reduce the size of Steiner tree instances by removing trivial parts [10, 27, 28]. While the Steiner tree B&B algorithms imply B&B algorithms for the general Steiner forest case, the Steiner *forest* problem was mostly studied in the context of approximation algorithms [1, 15, 17, 18, 20]. A PTAS on planar and bounded treewidth graphs exists [3].

Notation. Throughout, let $G = (V, E)$ be an undirected, simple graph and let $A = \{(i,j),(j,i) \mid \{i,j\} \in E\}$ be the arcs of the bidirection of G. A *cut-set* in G is a subset $S \subseteq V$. Any cut-set $S \subseteq V$ induces a *cut* $\delta(S) := \{\{i,j\} \in E \mid |\{i,j\} \cap S| = 1\}$. We abbreviate $\delta(i) := \delta(\{i\})$ if $S = \{i\}$. If $D = (V, A)$ is a directed graph, we distinguish the *outgoing cut* $\delta^+(S) = \{(i,j) \in A \mid i \in S \text{ and } j \notin S\}$ and the *incoming cut* $\delta^-(S) = \{(i,j) \in A \mid i \notin S \text{ and } j \in S\}$. Given a vector $x \in X^d$, $d \in \mathbb{Z}_{\geq 0}$, and an index set $I \subseteq \{1, \dots, d\}$ we write $x(I)$ to abbreviate $\sum_{i \in I} x_i$. Finally, for $k \in \mathbb{Z}_{\geq 1}$, let $[k] := \{1, \dots, k\}$.

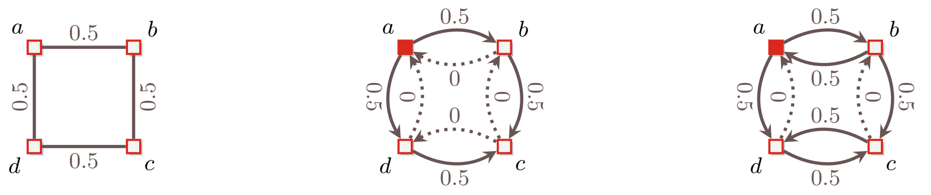
The Steiner forest problem. Consider an undirected graph $G = (V, E)$ together with $K \in \mathbb{N}$ terminal sets $T^1, \dots, T^K \subseteq V$. A *feasible* Steiner forest is a forest $(V_F \subseteq V, E_F \subseteq E)$ in G that, for all $k \in [K]$, contains an s-t-path for all $s, t \in T^k$. A feasible forest (V_F, E_F) is optimum with respect to edge weights $c \in \mathbb{R}^{|E|}_{\geq 0}$ if it minimizes the total cost $\sum_{e \in E_F} c_e$. Assume without loss of generality that the terminal sets are pairwise disjoint: If T^k and T^ℓ share at least one node, then any forest is feasible for $T^1, \dots, T^K$ if and only if it is feasible for the instance where T^k and T^ℓ are replaced by $T^k \cup T^\ell$. We denote the set of all terminal nodes by $\mathfrak{T} := T^1 \cup \dots \cup T^K$ and write $\tau(t) := k$ if $t \in T^k$. For each terminal set T^k, $k \in [K]$, we select an arbitrary node $r^k \in T^k$ as a fixed root node and define $\mathfrak{R} := \{r^1, \dots, r^K\}$. A cut-set $S \subseteq V$ is relevant for the terminal set T^k if it separates r^k from some terminal $t \in T^k$, i.e., if $r^k \in S$ but $t \notin S$ for some $t \in T^k$. We write $\mathfrak{S}^k$ for the set of all cut-sets that are relevant for T^k and $\mathfrak{S} := \mathfrak{S}^1 \cup \dots \cup \mathfrak{S}^K$ for the set of all relevant cut-sets. If $P := \{(x,y) \in \mathbb{R}^{n_1+n_2} \mid Ax + By = d\}$ is a polyhedron let $\operatorname{Proj}_x(P) := \{x \in \mathbb{R}^{n_1} \mid \exists\, y \in \mathbb{R}^{n_2} : (x,y) \in P\}$ be the projection of P onto the x variables.

2 Eliminating cycles from the linear programming relaxation

Let us briefly review the existing branch-and-bound algorithms and how they model the Steiner forest problem as an ILP. A forest F in $G = (V, E)$ is feasible if and only if any relevant cut-set $S \subset V$ contains at least one edge of F, i.e. if $|\delta_F(S)| \geq 1$ for all $S \in \mathfrak{S}$. Thus, since $c \geq 0$, the undirected cut formulation

$$\min\{c^{\mathrm{T}}x \mid x \in \mathfrak{L}^{uc} \text{ and integer}\} \text{ where} \tag{IPuc}$$
$$\mathfrak{L}^{uc} := \{x \in [0,1]^E \mid x(\delta(S)) \geq 1 \quad \forall S \in \mathfrak{S}\} \tag{1}$$

is a valid ILP formulation. While it can be solved efficiently, it yields weak bounds even on trivial instances (see Figure 1). The reason for the weak bounds becomes apparent when we see formulation (IPuc) as a set cover problem: We look for a choice of edges such that each cut $\delta(S)$ in G is covered by at least one edge. Consider any cycle C of length s in G. Any set cover needs $s - 1$ edges to cover C. On the other hand, we obtain a fractional solution of value $\frac{s}{2}$ by setting $x_e = 0.5$ for all edges $e \in C$. Figure 2a shows an example.



(a) A feasible solution for (1). The edges $\{c,d\}$ and $\{b,c\}$ cover the cuts $\delta(\{a,b,c\})$ and $\delta(\{a,b,d\})$.

(b) An *infeasible* solution for (2a)–(2d). The *arcs* (d,c) and (b,c) cover the cut $\delta(\{a,b,d\})$, but not the cut $\delta(\{a,b,c\})$.

(c) A feasible solution for (2a)–(2d). Additional capacity is needed on (c,d) and (b,a) to cover all relevant cuts.

Figure 2 An unit cost example where $\mathfrak{L}^{dc}$ yields a stronger LP bound than $\mathfrak{L}^{uc}$. The instance has a single terminal set that contains all four nodes of the graph. Node a has been chosen as the root.

The formulation can be improved with a standard construction [7, 2]. Recall that we choose $r^k \in T^k$ as an arbitrary root node of set T^k and consider the bi-directed graph underlying G. For all $k \in [K]$, we now look for an arborescence (a directed tree) rooted at r^k. If any cut-set S is relevant for T^k, then at least one arc must leave S:

$$\min\{c^{\mathrm{T}}x \mid (x,y) \in \mathfrak{L}^{dc} \text{ and integer}\} \text{ where} \tag{IPdc}$$

$$\mathfrak{L}^{dc} := \Big\{(x,y) \;\Big|\; y^k(\delta^+(S)) \geq 1 \qquad \forall\, k \in [K], \forall S \in \mathfrak{S}^k \tag{2a}$$

$$y^k_{ij} + y^k_{ji} \leq x_{ij} \qquad \forall\, \{i,j\} \in E, \forall k \in [K] \tag{2b}$$

$$y^k_{ij}, y^k_{ji} \in [0,1] \qquad \forall\, \{i,j\} \in E, \forall k \in [K] \tag{2c}$$

$$x_{ij} \in [0,1] \qquad \forall\, \{i,j\} \in E \Big\}. \tag{2d}$$

Since any solution (x,y) of (IPdc) can be turned into a feasible Steiner forest $F := \{\{i,j\} \in E \mid \exists\, k : y^k_{ij} + y^k_{ji} \geq 1\}$ and any feasible Steiner forest can be turned into a solution to (IPdc), this strengthened formulation indeed captures the Steiner forest problem. The formulation eliminates directed cycles from the basic optima of its LP relaxation and indeed the bound of the relaxation coincides with the integer optimum on instance A from Figure 1. However, a slightly modified instance makes the problem reappear, see instance B in Figure 1 or Figure 3: While the support of any y^k is free of directed cycles, the *union* of the supports is not. This is the reason why the formulation works exceptionally well for the Steiner *tree* problem where $K = 1$. If $K > 1$, however, the LP relaxation of (IPdc) is again weak. Still, for practical purposes no better formulation was known prior to this work. The offending cycles potentially appear whenever two terminal sets T^k and T^ℓ – and thus their roots r^k and r^ℓ – end up in the same connected component of the solution, i.e. of the support of x. If we knew beforehand that T^k and T^ℓ lie in the same connected component of an optimum solution, we could simplify the instance, replacing T^k and T^ℓ by their union $T^k \cup T^\ell$. Iterating this idea would yield a solution where all the arborescences are disjoint and the offending cycles are eliminated.

Unfortunately, we cannot know the connected components of a Steiner forest a priori. Instead, Magnanti and Raghavan [25] – we denote their model by (IPmr) and the LP relaxation by $\mathfrak{L}^{mr}$ – propose to compute the connected components of a solution on-the-fly in the ILP formulation. Then, whenever T^k and T^ℓ, $k \leq \ell$, lie in the same connected component, they look for a common arborescence that is rooted at r^k and connects all terminals in $T^k \cup T^\ell$. Unfortunately, their formulation has a size of $\Omega(\prod_{k=1}^{K} \sum_{\ell=k}^{K} |T^\ell|)$, i.e. it is exponential in the number of terminal sets K. We shall see in the next section how we achieve the same effect with a much smaller ILP formulation.

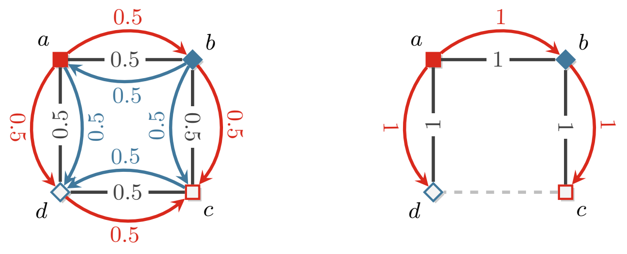


Figure 3 Detailed picture of instance (B) from Figure 1. *On the left:* The red and blue arcs form a solution for relaxation (2a)–(2d) for the red (□) and blue (◇) terminal set. The gray edges show the values of the x variables. Looking for a Steiner arborescence for each terminal set does not cut off a fractional optimum of cost 2. *On the right.* A solution that roots different terminal sets at the root node of the red (□) terminal set. The fractional optimum is cut off.

3 An new ILP formulation for the Steiner forest problem

Our extended formulation makes use of three kinds of variables. As before, we use a variable x_{ij} for all edges $\{i,j\} \in E$ to determine if $\{i,j\}$ is included in the forest F and two corresponding directed variables y_{ij}, y_{ji}. Likewise, the variables y^k_{ij} and y^k_{ji} for each $k \in [K]$ and each $\{i,j\} \in E$ determine if the arcs (i,j) and (j,i), respectively, are included in the arborescence rooted at r^k. Finally, we introduce an additional variable $z_{k\ell}$ for each $k \in [K]$ and each $\ell \geq k$, with the interpretation that $z_{k\ell} = 1$ iff T^k and T^ℓ both lie in the arborescence spanned by y^k. In the latter case, we say that r^k is responsible for the terminals in T^ℓ. To make it easier to state the formulation, we define $\mathfrak{T}^{i\ldots j}$ as $T^i \cup \cdots \cup T^j$ and let $\mathfrak{T}^{i\ldots j}_r := \mathfrak{T}^{i\ldots j} \setminus \{r^i\}$ be the same set without the ith root node (all other root nodes are still included). In particular, the set $\mathfrak{T}^{\ell\ldots K}_r$ contains all the terminal nodes that can potentially be connected to r^ℓ. We extend our previous notion and say that a cut-set $S \subseteq V$ is relevant for r^k and T^ℓ if $r^k \in S$ and some terminal $t \in T^\ell$ is not in S. The set of all cut-sets that are relevant for r^k and T^ℓ is written by $\mathfrak{S}^k_\ell$ in the sequel. Then, our formulation reads:

$$\min\left\{c^{\mathrm{T}}x \;\middle|\; (x,y,z) \in \mathfrak{L}^{sedc} \text{ and integer}\right\} \text{ where} \qquad \text{(IPsedc)}$$

$$\begin{aligned}
\mathfrak{L}^{sedc} := \Big\{(x,y,z) \;\Big|\; y^k(\delta^+(S)) &\geq z_{k\ell} && \forall\, k \in [K], \ell \geq k, \forall S \in \mathfrak{S}^k_\ell && \text{(3a)}\\
\sum_{\ell=1}^{k} z_{\ell k} &= 1 && \forall\, k \in [K] && \text{(3b)}\\
y_{ij} \geq \sum_{k\in[K]} y^k_{ij},\; y_{ji} &\geq \sum_{k\in[K]} y^k_{ji} && \forall\, \{i,j\} \in E && \text{(3c)}\\
z_{kk} &\geq z_{k\ell} && \forall\, k \in [K]\setminus\{1,K\}, \forall \ell \geq k+1 && \text{(3d)}\\
y_{ij} + y_{ji} &\leq x_{ij} && \forall\, \{i,j\} \in E && \text{(3e)}\\
y(\delta^-(v)) &\leq 1 && \forall\, v \in V && \text{(3f)}\\
y^k(\delta^-(t)) &= 0 && \forall\, k \in [K]\setminus\{1\}, \forall\, t \in \mathfrak{T}^{1\ldots k-1} && \text{(3g)}\\
y^k_{ij}, y^k_{ji} &\in [0,1] && \forall\, \{i,j\} \in E, \forall k \in [K] && \text{(3h)}\\
x_{ij}, y_{ij}, y_{ji} &\in [0,1] && \forall\, \{i,j\} \in E && \text{(3i)}\\
z_{k\ell} &\in [0,1] && \forall\, k \in [K], \forall \ell \geq k\Big\}. && \text{(3j)}
\end{aligned}$$

For any k, ℓ, the left hand side of the directed cut-set constraint (3a) is non-negative and the constraint is trivially satisfied if $z_{k\ell} = 0$. If otherwise $z_{k\ell} = 1$, we need to connect all terminals from T^ℓ to the k-th root r^k. Then, any cut-set S separating r^k from some terminal in T^ℓ must have at least one outgoing edge. This is exactly the condition modeled by (3a). For each $k \in [K]$, the constraints (3b) ensure that exactly one root r^ℓ is responsible for T^k (and r^1 is always responsible for T^1, i.e., $z_{11} = 1$). We use constraints (3c) to enforce that each edge $\{i, j\}$ is part of at most one arborescence. We also want to make sure that no "transitive" responsibilities exist: If r^k is responsible for T^ℓ, then r^ℓ cannot be responsible for some T^m, $m \neq \ell$. This is modeled by the symmetry breaking constraints (3d). They make sure that if root r^k is responsible for some terminal set T^ℓ, then r^k must be responsible for T^k as well. The capacity constraints (3e) say that if an edge $\{i, j\}$ is used in any arborescence, then it must be included in the tree. Moreover, no node in any arborescence should have more than one incoming arc, as modeled by the indegree constraints (3f). Finally, the terminals in $T^{1 \ldots k-1}$ cannot be attached to root r^k and thus, no arc of the corresponding arborescence should enter such a terminal, see constraint (3g).

To solve $\mathfrak{L}^{sedc}$ efficiently, we only add a subset of the cut-set constraints (3a) at the beginning. Then, given a solution (x^*, y^*, z^*) to a partially generated $\mathfrak{L}^{sedc}$, we can find a relevant S, a k and an ℓ such that $y^k(\delta^+(S)) < z_{k\ell}$ (or decide that none exist) efficiently: For each $k \in [K]$, each $\ell \geq k$ and each $t \in T^\ell$, we compute a minimum r^k-t-cut. If for any k, ℓ such a cut $\delta(S)$ has a value of strictly less than z_{kl}, then (x^*, y^*, z^*) does not satisfy the cut-set constraint corresponding to S and we add it to our LP and iterate. Otherwise, *all* cuts $S \in \mathfrak{S}^k_\ell$ must have a value of at least $z_{k\ell}$ and $(x^*, y^*, z^*) \in \mathfrak{L}^{sedc}$.

▶ **Lemma 1.** *Formulation (IPsedc) models the Steiner forest problem correctly. Its LP relaxation $\mathfrak{L}^{sedc}$ can be solved in time polynomial in the size of G and K.*

Strength of the new formulation. How can we compare two LP relaxations $\mathfrak{L}^A$ and $\mathfrak{L}^B$? We cannot expect that the bound from $\mathfrak{L}^A$ is stronger than the bound from $\mathfrak{L}^B$ on all instances: Generally, the optima of both LPs will be integral on *some* instances and must coincide then. We can, however, ask that the bound obtained from $\mathfrak{L}^A$ is never worse than the bound obtained from $\mathfrak{L}^B$. This is the case if any solution to $\mathfrak{L}^A$ is feasible for $\mathfrak{L}^B$ as well, i.e. if $\mathfrak{L}^A \subseteq \mathfrak{L}^B$. We say that $\mathfrak{L}^A$ is *strictly stronger* than $\mathfrak{L}^B$ if additionally at least one solution of $\mathfrak{L}^B$ is infeasible for $\mathfrak{L}^A$, i.e. if $\mathfrak{L}^A \subsetneq \mathfrak{L}^B$. In general, some truncation or extension of the solution might be necessary if $\mathfrak{L}^A$ and $\mathfrak{L}^B$ live in different variable spaces, but we can project the solutions suitably.

Instead of comparing the models directly, we compare their equivalent flow-based models; replacing the cut-condition by a flow-balance constraint. We also introduce additional flow variables f. Any feasible solution to $\mathfrak{L}^{sedf}$ defines a flow $f^{k,t}$ from r^k to any terminal $t \in \mathfrak{T}^{k \ldots K}$ and ensures that the flow value of $f^{k,t}$ is exactly $z_{k\ell}$.

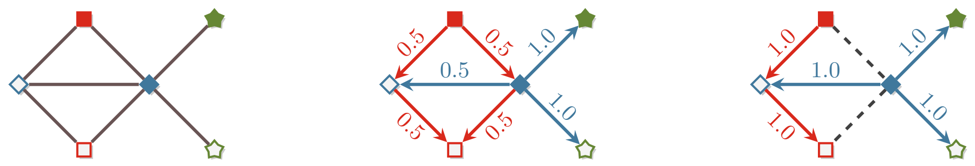
$$\mathfrak{L}^{sedf} := \Big\{ (x, y, f, z) \;\Big|\; f^{kt}_{ij} \leq y^k_{ij}, f^{kt}_{ji} \leq y^k_{ji} \qquad \begin{array}{l} \forall k \in [K], \forall \{i,j\} \in E \\ \forall t \in \mathfrak{T}^{k \ldots K}_r \end{array} \tag{4a}$$

$$f^{kt}(\delta^+(i)) - f^{kt}(\delta^-(i)) = \sigma_{kt}(i) z_{k\tau(t)} \qquad \begin{array}{l} \forall\, i \in V, \forall k \in [K] \\ \forall t \in \mathfrak{T}^{k \ldots K}_r \end{array} \tag{4b}$$

$$f^{kt}(\delta^+(t)) = 0 \qquad \forall k \in [K], \forall t \in \mathfrak{T}^{k \ldots K}_r \tag{4c}$$

$$\text{(3b)–(3j)} \tag{4d}$$

$$f^{kt}_{ij}, f^{kt}_{ji} \in [0,1] \qquad \begin{array}{l} \forall k \in [K], \forall t \in \mathfrak{T}^{k \ldots K}_r \\ \forall \{i,j\} \in E \end{array} \Big\}. \tag{4e}$$



(a) Instance with three terminal sets (□ 1, ◇ 2, ☆ 3) and unitary edge costs 1.

(b) Optimum solution of $\mathfrak{L}^{mr}$ with overall cost 4.5. This solution is infeasible for $\mathfrak{L}^{sedc}$ since here we would have $z_{22} = 0.5$ and $z_{23} = 1.0$, conflicting (3d).

(c) Optimum solution of $\mathfrak{L}^{sedc}$ which is integer and has cost 5. Here, non-0 z variables are $z_{11} = z_{22} = z_{23} = 1.0$.

Figure 4 Example instance where $\mathfrak{L}^{sedc}$ gives a stronger bound than $\mathfrak{L}^{mr}$.

The constant $\sigma_{kt}(i)$ is set to 1 if $i = r^k$, to -1 if $i = t$, and to 0 otherwise. The constraints (4c) prohibit f^{kt} from leaving t and facilitate the comparison to $\mathfrak{L}^{mr}$. Analogously, the relaxation $\mathfrak{L}^{dc}$ has an arc-flow-based equivalent $\mathfrak{L}^{df}$ that forces a choice of arcs such that each root r^k is able to send one unit of flow to each terminal in $T^k \setminus \{r^k\}$.

▶ **Theorem 2.** $\text{Proj}_x(\mathfrak{L}^{sedc}) \subsetneq \text{Proj}_x(\mathfrak{L}^{dc})$

Proof sketch. We prove the claim by showing that $\mathfrak{L}^{sedf}$ is strictly stronger than $\mathfrak{L}^{df}$. Let thus $(x, y, f, z) \in \mathfrak{L}^{sedf}$. We want to show that there exists some y' and some flow f' such that $(x, y', f') \in \mathfrak{L}^{df}$. Here, the challenge is that there might be a non-zero flow $f^{k,t}$ from r^k to $t \in T^\ell$ whereas in $\mathfrak{L}^{df}$, all the flow to t must originate from r^ℓ. Still, we can morally obtain a feasible flow f' in the following way: Since $r^\ell \in T^\ell$, there must be a flow of value exactly $z_{k\ell}$ from r^k to r^ℓ. But r^k also sends a flow with value $z_{k\ell}$ to t. Thus, if we reverse f^{k,r^ℓ} we can concatenate it with $f^{k,t}$ and maintain flow conservation. We remove all cycles and we obtain the desired flow f'. However, this construction might force us to change the orientation of some of the arcs, which poses an additional technical difficulty. Finally, we can iterate this argument and combine all flows $f^{m,t}$ from any root r^m to t. By constraint (3b), these flows must add up to one. Strictness follows from instance (B) in Figure 1. ◀

Our second theoretical result is that the new relaxation $\mathfrak{L}^{sedc}$ is strictly stronger than the relaxation of [25]. Due to space restrictions we refer the reader to [25] for the description of $\mathfrak{L}^{mr}$ with constraints (14b)–(14j).

▶ **Theorem 3.** $\text{Proj}_x(\mathfrak{L}^{sedc}) \subsetneq \text{Proj}_x(\mathfrak{L}^{mr})$

Proof sketch. As before, we compare $\mathfrak{L}^{sedf}$ instead of $\mathfrak{L}^{sedc}$. The major difference between $\mathfrak{L}^{sedf}$ and $\mathfrak{L}^{mr}$ is this: While in $\mathfrak{L}^{sedf}$, any two flows f^{kt} and $f^{kt'}$ for $t, t' \in T^\ell$ must have the same flow value $z_{k\ell}$, the same flows can have different values in $\mathfrak{L}^{mr}$. In that sense, $\mathfrak{L}^{sedf}$ is more restricted and it makes sense that any flow that is feasible in $\mathfrak{L}^{sedf}$ is feasible in $\mathfrak{L}^{mr}$, too, whereas the converse is not necessarily true (see Figure 4). More formally, let $(\bar{x}, \bar{y}, \bar{z}, \bar{f}) \in \mathfrak{L}^{sedf}$. We argue that $(\bar{x}, \bar{y}, \bar{f}) \in \mathfrak{L}^{mr}$. It follows from (4b) that $(\bar{x}, \bar{y}, \bar{f})$ satisfies (14b) from [25]. Constraint (14c) follows from (4b), (4c), and (3b). For (14d), apply (4b), (4b) with $t = \bar{\ell}$, and (4c). Constraint (14e) follows from (4a), (3c), and (3e). Likewise, constraint (14f) follows from (4a), (3c), and applying (3f). Finally, the constraint (14g) is implied by (3g). (14h) is equivalent to (4c). ◀

3.1 A smaller cut-based formulation

We remark that (IPsedc) can be written in the slightly different form below. While the reformulation is smaller and less involved, it turns out that its linear programming bounds are potentially weaker than the ones from (IPsedc). We need two variables y_{ij}, y_{ji}, and a variable x_{ij} for each edge $\{i,j\} \in E$. As before, for all $k \in [K]$ and all $\ell \geq k$, we have a decision variable $z_{k\ell}$ that tells us whether the terminals in T^ℓ should be connected to the root r^k.

$$\min\left\{c^{\mathrm{T}} x \;\middle|\; (x,y,z) \in \mathfrak{L}^{edc} \text{ and integer}\right\} \text{ where} \tag{IPedc}$$

$$\begin{aligned}
\mathfrak{L}^{edc} := \Bigg\{ (x,y,z) \;\Bigg|\; y(\delta^+(S)) &\geq \sum_{\substack{k \leq \ell:\\ r^k \in S}} z_{k\ell} && \forall \ell \in [K], \forall S \subseteq V \colon T^\ell \cap S \neq T^\ell && \text{(5a)}\\
\sum_{\ell=1}^{k} z_{\ell k} &= 1 && \forall k \in [K] && \text{(5b)}\\
z_{kk} &\geq z_{k\ell} && \forall k \in [K] \setminus \{1, K\}, \forall \ell \geq k+1 && \text{(5c)}\\
y_{ij} + y_{ji} &\leq x_{ij} && \forall \{i,j\} \in E && \text{(5d)}\\
y_{ij}, y_{ji}, x_{ij} &\in [0,1] && \forall \{i,j\} \in E && \text{(5e)}\\
z_{k\ell} &\in [0,1] && \forall k \in [K], \forall \ell \geq k \quad \Bigg\}. && \text{(5f)}
\end{aligned}$$

To see why the formulation is correct, consider a cut-set $S \subseteq V$ with $t \notin S$ for some terminal $t \in T^\ell$. If S contains a root node r^k with $z_{k\ell} = 1$, then S must have at least one outgoing arc and the right-hand side of (5a) evaluates to 1 (observe that because of (5b) the right-hand side can never exceed 1). Otherwise, the right-hand side of (5a) evaluates to 0 and the constraint is trivially satisfied. The LP relaxation of (IPedc) can be solved in polynomial time using a similar algorithm as for $\mathfrak{L}^{sedc}$.

▶ **Lemma 4.** $\mathrm{Proj}_x(\mathfrak{L}^{sedc}) \subsetneq \mathrm{Proj}_x(\mathfrak{L}^{edc})$.

Proof. Let $(\tilde{x}, \tilde{y}, \tilde{z}) \in \mathfrak{L}^{sedc}$. We argue that $(\tilde{x}, \tilde{y}, \tilde{z}) \in \mathfrak{L}^{edc}$. The constraints (5b)–(5d) are trivially satisfied. Now, consider a directed cut $S \subseteq V \colon S \cap T^\ell \neq \emptyset$, for some set $\ell \in [K]$. Any cut S is relevant to the sum in the right-hand side of constraint (5a) if and only if it is a valid cut for constraint (3a), hence

$$\tilde{y}(\delta^+(S)) \overset{(3c)}{\geq} \sum_{k=1}^{K} \tilde{y}^k(\delta^+(S)) \geq \sum_{k=1}^{\ell} \tilde{y}^k(\delta^+(S)) \overset{(3a)}{\geq} \sum_{k \leq \ell} \tilde{z}_{k\ell} \geq \sum_{\substack{k \leq \ell:\\ r^k \in S}} \tilde{z}_{k\ell}$$

and thus (5a) is satisfied. Again, strictness follows from instance (B) in Figure 1. ◀

On the other hand, the model is stronger than the directed model without z variables.

▶ **Lemma 5.** $\mathrm{Proj}_x(\mathfrak{L}^{edc}) \subsetneq \mathrm{Proj}_x(\mathfrak{L}^{dc})$.

We summarize the results of the discussion in Figure 5 and remark that the relationship of $\mathfrak{L}^{mr}$ to the models $\mathfrak{L}^{dc}$ and $\mathfrak{L}^{edc}$ is an open problem. Our conjecture is that it holds $\mathrm{Proj}_x(\mathfrak{L}^{mr}) \subsetneq \mathrm{Proj}_x(\mathfrak{L}^{edc}) \subsetneq \mathrm{Proj}_x(\mathfrak{L}^{dc})$.

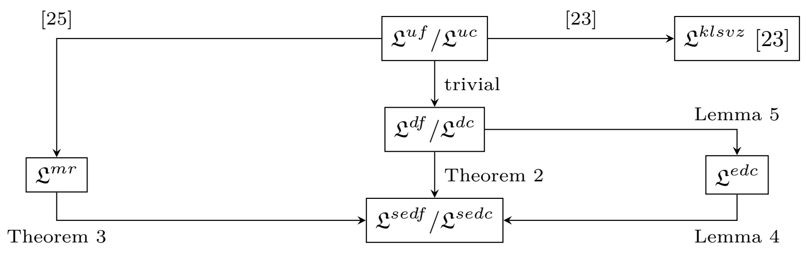

[25]
[23]
$\mathfrak{L}^{uf}/\mathfrak{L}^{uc}$
$\mathfrak{L}^{klsvz}$ [23]
trivial
$\mathfrak{L}^{df}/\mathfrak{L}^{dc}$
Lemma 5
$\mathfrak{L}^{mr}$
Theorem 2
$\mathfrak{L}^{edc}$
$\mathfrak{L}^{sedf}/\mathfrak{L}^{sedc}$
Theorem 3
Lemma 4

Figure 5 Relationship of the LP relaxations. The arrows point to the stronger relaxation.

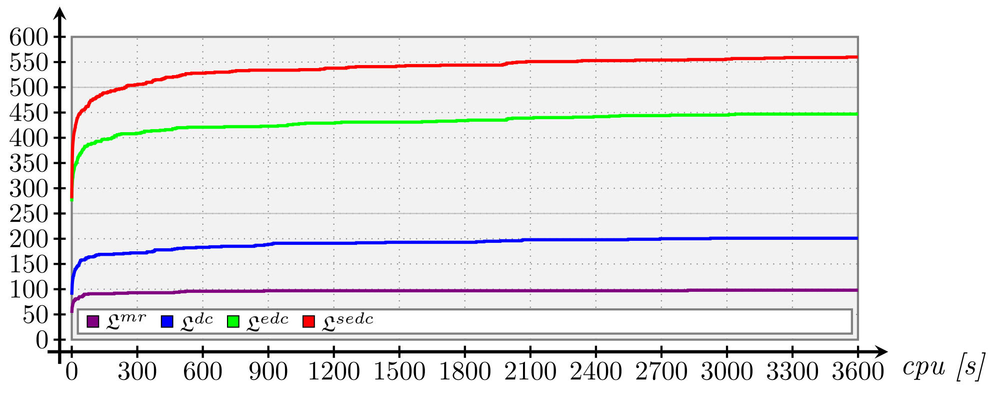


Figure 6 Number of `JMP` instances (out of 580) solved by B&B after x seconds.

4 Experimental results

Settings. All experiments were performed on a Debian 9.4 machine with an `Intel(R) Xeon(R) CPU E5-2643` running at 3.30GHz. Our code is written in `C++` using the `ILOG CPLEX 12.6.3` framework. We compiled with `gcc-6.3` and `-O2` flags. Automatic symmetry breaking and presolving was disabled in `CPLEX`, as well as all general integer cuts.

Instances. For the `JMP` instance set, we generated 580 random graphs with a frequently used method by Johnson, Minkoff, and Philipps [21]: First, distribute n nodes uniformly at random in a unit square. Then, insert an edge $\{i,j\}$ if the Euclidean distance between i and j is less than $\alpha/\sqrt{n}$, where α is a parameter for the random generator. The cost of the edge $\{i,j\}$ is proportional to the Euclidean distance. Finally, connect all nodes with a minimum Euclidean spanning tree to ensure that the instance is connected.

To determine K random terminal sets, we first select $t \cdot |V|$ nodes uniformly at random (the number $K \in [n/2]$ of terminal sets and the terminal percentage $t \in [0,1]$ are again parameters). We then bring the selected nodes into a random order and draw $K-1$ distinct split points from $\{2,\ldots,t\cdot|V|-1\}$, thus splitting the random node order into K distinct terminal sets. For each $n \in \{25,50,150,200,500\}$, we choose a small, a medium, and a large number of terminal sets K. The percentage t of terminal nodes is picked from $\{0.25,0.5,0.75,1.0\}$ unless a combination of n, K, and t results in a terminal set size of less than two. For each choice of n, K, and t, we generate five instances with $\alpha = 1.6$ and five instances with $\alpha = 2.0$; leading to 580 `JMP` instances. The `MR` instance set is generated based on [25] and contains 85 instances.

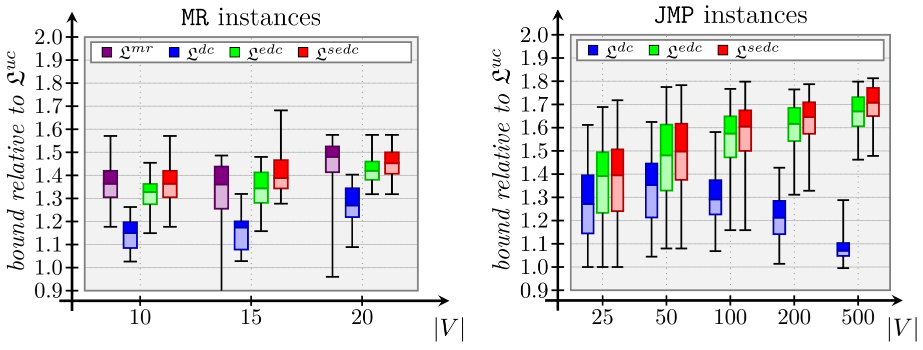


Figure 7 Improvement of the linear programming bound over the bound obtained from $\mathfrak{L}^{uc}$: The plot shows the ratio of the best bound after 1200 seconds over the optimum bound from $\mathfrak{L}^{uc}$. The theoretical maximum improvement is at most 2. *On the right:* For $|V| \in \{25, 50\}$ all bounds are optimum; for $|V| \in \{100, 200\}$ only the $\mathfrak{L}^{edc}$ and $\mathfrak{L}^{sedc}$ bounds are optimum. For $|V| = 500$, about 50% of the $\mathfrak{L}^{edc}$ and $\mathfrak{L}^{sedc}$ and none of the $\mathfrak{L}^{dc}$ bounds are optimum.

The branch-and-bound algorithm. We solve formulation (IPsedc) by an B&B algorithm. As the algorithm requires solving the LP relaxation $\mathfrak{L}^{sedc}$ in each B&B node, we generate the LP relaxation dynamically with the separation procedure from Section 3. This allows us to efficiently solve each B&B node. We follow the same approach for the ILP from [25], for (IPedc), and for (IPdc) and compare the results. Similar separation procedures using minimum cuts exist for $\mathfrak{L}^{dc}$ and $\mathfrak{L}^{edc}$, so that we can generate the relaxations efficiently as well. In more detail, we compute a minimum s-t-cut by computing a maximum s-t-flow f and then deriving a cut-set S, where a node $v \in V$ is included in S if and only if there is a directed path from v to t in the residual network of f. We can then derive a cut-set inequality based on S. Some algorithmic techniques have the potential to improve this on-the-fly generation [22]:

Back cuts. Additionally add the cut-set inequality corresponding to $\bar{S}$ where $v \in V$ is included in $\bar{S}$ if and only if there is a directed s-v-path in the residual network of f.

Nested cuts. Assign an infinite capacity to all saturated edges in the residual network of f and iterate. Nested cuts can be combined with back cuts: We first compute S and $\bar{S}$ and then compute nested cuts on both sets.

Creep flows. Add a small $\varepsilon = 10^{-8}$ to all capacities. This lets us find a minimum weight cut that cuts few edges. The creep flow variant works together with both nested cuts and back cuts.

Cut purging. Finally, it can be beneficial to remove cut-set inequalities from the relaxation if they have not been binding for a number of iterations.

It is not clear a priori which combination of these variants leads to the best performance of the algorithm. In a preliminary experiment, we evaluated all 16 combinations for all the formulations under consideration. To avoid overfitting, we tested on a random subset of the instances only. Back cuts were beneficial in all cases. The $\mathfrak{L}^{sedc}$ relaxation benefited from additional creep flows, while $\mathfrak{L}^{dc}$ worked best with additional nested cuts and purging. In all cases, we compute the maximum s-t-flows with a custom implementation of the push-relabel algorithm with the *highest-label* strategy and the *gap heuristic* [16, 5]. Since the $\mathfrak{L}^{sedc}$ constraints (3f) and (3g) would be valid for $\mathfrak{L}^{dc}$ and $\mathfrak{L}^{edc}$ as well, we compare against $\mathfrak{L}^{sedc}$ without (3f) and (3g). This results in a fairer comparison.

Comparison of the algorithms. We compare B&B algorithms based on the previous best formulations with our new ones in Figure 6 using the `JMP` instance set. All algorithms are run in the tuned configuration from the preliminary experiment. The figure shows that our new $\mathfrak{L}^{sedc}$ based algorithm solves almost twice as many instances to optimality as the previous ones. A more detailed picture would show that all unsolved instances are large ones with $|V| = 500$. The $\mathfrak{L}^{edc}$ based algorithm solves significantly less instances, but still performs better than the algorithm based on $\mathfrak{L}^{dc}$. The algorithm based on $\mathfrak{L}^{mr}$ mostly solved the small instances. The $\mathfrak{L}^{sedc}$, the $\mathfrak{L}^{edc}$, the $\mathfrak{L}^{dc}$, and the $\mathfrak{L}^{mr}$ based algorithm solved 480, 385, 185, and 97 instances without branching, respectively. In the following, we analyze why our algorithms perform well.

The new bounds can be computed quickly. We compare our new approach against the two previous best: The relaxation $\mathfrak{L}^{dc}$ of the directed cut formulation and the relaxation $\mathfrak{L}^{mr}$ of [25]. Ideally, we would like to have relaxations that solve quickly and yield a strong bound. Indeed, the LP relaxations in our new algorithm can be solved to optimality quickly and reliably: We find the LP optimum of at least 500 out of 580 instances within 1200 seconds. Moreover, the bulk of the LP relaxations is solved within 100 seconds for $\mathfrak{L}^{edc}$ and within 400 seconds for $\mathfrak{L}^{sedc}$. At the same time, the existing B&B algorithms struggle to solve their LP relaxations: The relaxation $\mathfrak{L}^{mr}$ could only be solved to optimality within 1200 seconds in 100 out of 580 times. In the same time frame, the relaxation $\mathfrak{L}^{dc}$ could be solved 280 times.

The new bounds are strong. The results from the previous section show that the optimum bound from the LP relaxations of the advanced formulations will never be worse than the bound from $\mathfrak{L}^{uc}$. We would like to quantify the ratio of the bounds; however, the theoretical worst-case ratio of the bounds is 1. What ratio can we hope for on non-artificial instances? To answer this question, we solve the LP relaxations of the advanced formulations. Since any feasible solution to an LP relaxation yields a valid bound, we stop the computation after 1200 seconds and take the best bound obtained up to that point. We then compare this bound with the optimum of $\mathfrak{L}^{uc}$ in Figure 7. The bounds obtained from the relaxation of [23] were exactly the same as of $\mathfrak{L}^{uc}$ and are not shown here. $\mathfrak{L}^{mr}$ only solved a significant number of the small instances from the `JMP` set. To nonetheless obtain a fair comparison for $\mathfrak{L}^{mr}$, we instead look at the `MR` instance set that is based on the original publication of [25]. The comparison can also be seen in Figure 7. We see that *if* $\mathfrak{L}^{mr}$ can be solved, it yields a bound that is comparable to the one from the new relaxations.

5 Conclusion

Overall, our new branch-and-bound algorithm works very well and its performance seems to be due to the strong bounds obtained from the new ILP formulation (IPsedc). While its relaxation $\mathfrak{L}^{sedc}$ is solved less quickly than the simplified relaxation $\mathfrak{L}^{edc}$, its stronger bounds seem to pay off overall. At the same time, it answers Magnanti's and Raghavan's open problem: There is indeed an equally strong cut-based model to [25]. On the theoretical side, we would like to obtain an LP relaxation with an integrality gap of much less than 2. This problem is not solved by $\mathfrak{L}^{sedc}$: We observe that it coincides with $\mathfrak{L}^{dc}$ if $K = 1$. On the other hand, Könemann et al. [23] propose an LP relaxation that has a better worst-case integrality gap. In our experiments, however, the relaxation always yields the same bounds as the weak undirected cut relaxation $\mathfrak{L}^{uc}$, making it less suitable for practical purposes.

References

1 A. Agrawal, P. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456, 1995.

2 A. Balakrishnan, T. L. Magnanti, and R. T. Wong. A dual-ascent procedure for large-scale uncapacitated network design. *Operations Research*, 37(5):716–740, 1989.

3 M. Bateni, M. T. Hajiaghayi, and D. Marx. Approximation schemes for Steiner forest on planar graphs and graphs of bounded treewidth. *Journal of the ACM*, 58(5):21:1–21:37, 2011.

4 A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: Fast subset convolution. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 67–74. ACM, 2007.

5 B. V. Cherkassky and A. V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997.

6 S. Chopra, E. R. Gorres, and M. R. Rao. Solving the Steiner tree problem on a graph using branch and cut. *ORSA Journal on Computing*, 4(3):320–335, 1992.

7 S. Chopra and M. R. Rao. The Steiner tree problem I: Formulations, compositions and extension of facets. *Mathematical Programming*, 64(1):209–229, 1994.

8 S. Chopra and M. R. Rao. The Steiner tree problem II: Properties and classes of facets. *Mathematical Programming*, 64(1-3):231–246, 1994.

9 S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

10 C. W. Duin and A. Volgenant. Reduction tests for the steiner problem in grapsh. *Networks*, 19(5):549–567, 2006.

11 J. Edmonds. Submodular functions, matroids, and certain polyhedra. In M. Jünger, G. Reinelt, and G. Rinaldi, editors, *Combinatorial Optimization — Eureka, You Shrink!*, number 2570 in LNCS, pages 11–26. Springer Berlin Heidelberg, 2003.

12 R. E. Erickson, C. L. Monma, and A. F. Veinott. Send-and-split method for minimum-concave-cost network flows. *Mathematics of Operations Research*, 12(4):634–664, 1987.

13 M. X. Goemans. The Steiner tree polytope and related polyhedra. *Mathematical Programming*, 63(1–3):157–182, 1994.

14 M. X. Goemans and Y.-S. Myung. A catalog of Steiner tree formulations. *Networks*, 23(1):19–28, 1993.

15 M. X. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

16 A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988.

17 M. Groß, A. Gupta, A. Kumar, J. Matuschke, D. R. Schmidt, M. Schmidt, and J. Verschae. A local-search algorithm for Steiner forest. In A. R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, volume 94 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

18 A. Gupta and A. Kumar. Greedy Algorithms for Steiner Forest. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 871–878. ACM, 2015.

19 S. Hougardy, J. Silvanus, and J. Vygen. Dijkstra meets Steiner: A fast exact goal-oriented Steiner tree algorithm. *Mathematical Programming Computation*, 9(2):135–202, 2017.

20 K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

21 D. S. Johnson, M. Minkoff, and S. Phillips. The prize collecting Steiner tree problem: Theory and practice. In *Proceedings of the Symposium on Discrete Algorithms*, SODA '00, pages 760–769. SIAM, 2000.

22 T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32(3):207–232, 1998.

23 J. Könemann, S. Leonardi, G. Schäfer, and S. van Zwam. A group-strategyproof cost sharing mechanism for the Steiner forest game. *SIAM Journal on Computing*, 37(5):1319–1341, 2008.

24 A. Lucena. Tight bounds for the Steiner problem in graphs. Technical report, RC for Process Systems Engineering, Imperial College, London, 1993.

25 T. L. Magnanti and S. Raghavan. Strong formulations for network design problems with connectivity requirements. *Networks*, 45:61–79, 2005.

26 F. Margot, A. Prodon, and T. M. Liebling. Tree polytope on 2-trees. *Mathematical Programming*, 63(1–3):183–191, 1994.

27 T. Polzin. *Algorithms for the Steiner Problem in networks.* PhD thesis, Universität des Saarlandes, 2004. URL: `http://scidok.sulb.uni-saarland.de/volltexte/2004/218/index.html`.

28 T. Polzin and V. S. Daneshmand. A comparison of Steiner tree relaxations. *Discrete Applied Mathematics*, 112(1):241–261, 2001.

Large Low-Diameter Graphs are Good Expanders

Michael Dinitz[1]
Dept. of Computer Science, Johns Hopkins University, Baltimore, US
mdinitz@cs.jhu.edu

Michael Schapira
School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel
schapiram@cs.huji.ac.il

Gal Shahaf
Dept. of Mathematics, The Hebrew University, Jerusalem, Israel
gal.shahaf@mail.huji.ac.il

Abstract

We revisit the classical question of the relationship between the diameter of a graph and its expansion properties. One direction is well understood: expander graphs exhibit essentially the lowest possible diameter. We focus on the reverse direction, showing that "sufficiently large" graphs of fixed diameter and degree must be "good" expanders. We prove this statement for various definitions of "sufficiently large" (multiplicative/additive factor from the largest possible size), for different forms of expansion (edge, vertex, and spectral expansion), and for both directed and undirected graphs. A recurring theme is that the lower the diameter of the graph and (more importantly) the larger its size, the better the expansion guarantees. Aside from inherent theoretical interest, our motivation stems from the domain of network design. Both low-diameter networks and expanders are prominent approaches to designing high-performance networks in parallel computing, HPC, datacenter networking, and beyond. Our results establish that these two approaches are, in fact, inextricably intertwined. We leave the reader with many intriguing questions for future research.

2012 ACM Subject Classification Mathematics of computing → Spectra of graphs, Mathematics of computing → Extremal graph theory, Networks → Network design principles, Networks → Network structure

Keywords and phrases Network design, Expander graphs, Spectral graph theory

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.71

Acknowledgements We wish to thank Nati Linial, Alex Samorodnitsky, Elchanan Mossel, and Noga Alon for fruitful discussions.

1 Introduction

Both the diameter of a graph and its expansion capture the "connectedness" of the graph, albeit in two very different senses. The diameter, i.e., the maximal distance between a pair of vertices, provides an upper bound on the length of shortest paths in the graph, whereas expansion measures the minimal ratio between a subset of vertices and its boundary. We revisit the classical question of relating these two traits. One direction is well known: good expansion implies a low diameter. Specifically, the diameter of a graph with good expansion

[1] [Supported in part by NSF awards 1464239 and 1535887]

© Michael Dinitz, Michael Schapira, and Gal Shahaf;
licensed under Creative Commons License CC-BY

is $O(\log n)$ (see, e.g., [26]), which is asymptotically the lowest possible. We focus on the opposite, and largely unexplored, direction.

In general, low diameter does not guarantee good expansion. Consider, e.g., a graph on n vertices that is a disjoint union of two cliques, each of size $\frac{n}{2}$. Removing one edge from each clique and connecting the cliques via two "bridges" results in a $(\frac{n}{2}-1)$-regular graph of diameter 3 with very low expansion (which worsens as $n \to \infty$). We observe, however, that this "bad" graph is significantly smaller than the largest $(\frac{n}{2}-1)$-regular graph of diameter 3 (which is of size $\Omega(n^3)$). Indeed, our investigation below reveals that, in contrast to the above, when the degree and the diameter are fixed and the size of the graph is "sufficiently large", the graph must have "good" expansion. We formalize this statement for different notions of "large", for different forms of expansion (edge, vertex, and spectral expansion), and for undirected/directed graphs. Our results are presented in Section 1.2, but informally, "sufficiently large" means that the size of the graph is close to the best-known upper bound on the size, in either a multiplicative or additive setting.

We formalize the above statements and discuss implications of our results for network design and beyond, including the unification of two competing approaches to datacenter network design.

1.1 The Degree/Diameter Problem

Before we can state our results, we must first define what we mean by a "large, low-diameter graph". To start off: how large can a d-regular graph of diameter k (which we shall refer to as a "(d,k)-graph" henceforth) actually be? An upper bound on the size of such a graph is the classical *Moore Bound* [25], denoted by $\mu_{d,k}$ (see Section 2 for a formal definition). Extensive research has been devoted to determining the existence of graphs whose sizes match this upper bound (a.k.a., Moore Graphs) or well-approximate it. This line of study, termed the *"degree/diameter problem"*, was initiated by Hoffman and Singleton [25]. See [41] for a detailed survey of results in this active field of research.

Graphs whose sizes exactly match the Moore Bound, referred to as "Moore Graphs" henceforth, only exist for very few values of d and k [25, 4]. Consequently, various constructions for generating graphs whose sizes come "close" to the Moore Bound, which we call "approximate-Moore Graphs", have been devised.

Specifically, constant multiplicative approximations (MMS-graphs [39]) and constant additive approximations (e.g., polarity graphs [20, 12]) have been devised for the case of diameter $k = 2$. Delorme [41, 16, 17] constructed infinite series of (d,k)-graphs whose sizes arbitrarily approach the Moore Bound for diameters $k = 3$ and $k = 5$. Graph constructions whose sizes approximate the Moore Bound within non-constant multiplicative factors exist for arbitrary values of k (examples include, e.g., *de Bruijn* [15] and *Canale-Gomez* [13] graphs for the undirected case, and *Alegre* and [41] *Kautz* [19] digraphs for the directed analogue of the problem). While constructing approximate-Moore Graphs whose sizes arbitrarily approach the Moore Bound for arbitrary values of k remains an important and widely studied open question, such graphs are believed to exist for sufficiently large d and k, as conjectured, e.g., by Bollobás in [10].

Our investigation of the relation between diameter and expansion also uses the Moore Bound as a benchmark and compares the size of (d,k)-graphs to $\mu_{d,k}$. We consider both multiplicative and additive approximations to the Moore Bound. Our results establish that good solutions to the degree-diameter problem *must* be good expanders, establishing a novel link between two prominent and classical lines of research. In addition, our results yield new expansion bounds for all of the classical constructions of low-diameter graphs discussed above.

Table 1 Summary of Results Relating Expansion and Diameter.

	Size	**Expansion guarantees**
(d,k)-graph	$n \geq \mu_{d,k} - O(d^{k/2})$	$\lambda(G) = O(\sqrt{d})$
	$n \geq (1-\varepsilon)\mu_{d,k}$	$\lambda(G) = O(\varepsilon^{1/k})d$
	$n = \alpha \cdot \mu_{d,k}$	$h_e(G) \geq \frac{\alpha d}{2k} \cdot \left(1 - \frac{1}{(d-1)^k}\right)$
		$\phi_V(G) \geq \frac{\alpha}{2(k-1)+\alpha}$
$k = 2$	n	$\lambda(G) \leq \frac{1+\sqrt{1+4(d^2+d-n)}}{2}$
	$n = \alpha \cdot d^2$	$h_e(G) \geq \frac{2d+1-\sqrt{4(1-\alpha)d^2+4d+1}}{4}$
		$\phi_V(G) \geq \frac{2\alpha}{2\alpha+1}$
$k = 3$	$n = \alpha \cdot d^3$	$\phi_V(G) \geq \frac{\alpha}{\alpha+1}$
(d,k)-digraph	$n = \alpha \cdot \widetilde{\mu_{d,k}}$	$h_e(G) \geq \frac{\alpha}{2k}(d - \frac{1}{d^k})$
		$\phi_V(G) \geq \frac{\alpha \cdot d}{2(d+1)(k-1)+\alpha \cdot d}$

1.2 Our Results

Our results relating the size of a (d,k)-graph to its expansion are summarized in Table 1, with $\lambda(G)$, $h_e(G)$, and $\phi_V(G)$ denoting spectral expansion, edge expansion, and vertex expansion, respectively (formal definitions can be found in Section 2). $\widetilde{\mu_{d,k}}$ is the analogue of the Moore Bound for directed graphs.

We begin in Section 3 with our main results, which provide bounds on the spectral expansion of large (d,k)-graphs. In particular, our results establish that if the size of a graph is very close *additively* to the Moore bound, then the graph is essentially an optimal expander. In addition, if the graph has size that is close *multiplicatively* to the Moore bound, the spectral expansion might no longer be optimal, but is still very good.

We next turn our attention to combinatorial notions of expansion: edge expansion and vertex expansion. We provide (in Section 4) guarantees on both the edge and the vertex expansion of (d,k)-graphs in terms of their multiplicative distance from the Moore Bound. Our analysis leverages careful counting arguments to bound the ratio between the cardinality of a set of vertices and the size of its boundary. We also prove, through more refined analyses, improved results for diameters 2 and 3.

The key technical insight underlying our results for spectral expansion is a novel link, which we believe is of independent interest, between the nontrivial eigenvalues of a graph's adjacency matrix and the distance of the graph from the Moore Bound. Specifically, the proofs of our results for spectral expansion rely on the analysis of *non-backtracking paths* in the graph. A path is said to be non-backtracking if it does not traverse an edge back and forth consecutively. We prove that the matrix that corresponds to all non-backtracking paths of length at most k must consist of strictly positive entries and shares all eigenvectors of the adjacency matrix A. We establish the above algebraic relation between the two matrices by employing the *Geronimus Polynomials* [9, 44], a well-known class of orthogonal polynomials, as operators acting on the adjacency matrix. Given the spectrum of A, an asymptotic estimation of the polynomials' coefficients allows us to bound the spectrum of the non-backtracking paths matrix. We then subtract from the latter the all-ones matrix and use the leading eigenvalue of the remaining matrix (which can be computed directly) to bound the nontrivial eigenvalues of the adjacency matrix A.

Table 2 Implications for Known Constructions of Low-Diameter Graphs.

Construction	Spectral expansion	Edge expansion	Vertex expansion
de Bruijn ($k=2$) [15]	-	–	$\frac{1}{3}$
Canale-Gomez [13]	-	$\frac{d}{2k\cdot 1.57^k}\left(1-\frac{1}{(d-1)^k}\right)$	$\frac{1.57^{-k}}{2(k-1)+1.57^{-k}}$
Alegre digraph [41]	-	$\frac{25\cdot 2^k}{32k\cdot d^k}\left(d-\frac{1}{d^k}\right)$	$\frac{\left(\frac{2}{d}\right)^k\cdot\frac{25d}{16}}{2(d+1)(k-1)+\left(\frac{2}{d}\right)^k\cdot\frac{25d}{16}}$
Kautz digraphs [19]	-	$\frac{1}{2k}\left(d-\frac{1}{d^k}\right)$	$\frac{d}{2(d+1)(k-1)+d}$
Polarity graph [20, 12]	$\lambda\leq\frac{1+\sqrt{1+8(d-1)}}{2}$	$\frac{2d+1-\sqrt{4d+1}}{4}$	$\frac{2}{3}$
MMS-graphs [39]	$\lambda\leq\frac{1+\frac{1}{3}\sqrt{d^2+d+7}}{2}$	$\frac{2d+1-\sqrt{\frac{4}{9}d^2+4d+1}}{4}$	$\frac{16}{25}$

Our technique should be contrasted with employing Hashimoto's non-backtracking operator [24] to reason about non-backtracking paths in a graph (e.g., in the context of localization and centrality [38], clustering [33], mixing time acceleration [1], and percolation [27] in networks). In our context, applying Hashimoto's operator involves reasoning about intricate relations between the spectra of the adjacency matrix A and another matrix, called the "non-backtracking matrix" (via the Ihara-Bass formula [11]). Geronimous Polynomials, a subfamily of the more renowned Chebyshev polynomials, allow for much simpler analysis. We believe that our techniques, and the (yet to be explored) connections between Geronimous Polynomials and Hashimoto's operator, are of independent interest and may find wider applicability.

Importantly, our research diverges from the main vein of prior research on expanders. Expanders are commonly viewed as highly-connected sparse graphs. Indeed, the bulk of literature on this topic assumes that the degree of these graphs is essentially constant with respect to the size of the graph (i.e., $d \ll n$). In contrast, the size of a "large" (d,k)-graph graph is $O(d^k)$.

Aside from inherent theoretical interest, our motivation stems from the domain of network design. Low-diameter networks have been widely studied in the context of high-performance-computing (HPC) architectures (see, e.g., [29, 8, 2, 30]), parallel computing [34], and the design of fault-tolerant networks [6, 7, 8, 21, 42]. Of special interest in this literature are large networks of very low diameters (e.g., 2 or 3), as short path lengths translate to low latency in data delivery and also to low packet-queueing delays and power consumption (due to having few intermediate network devices en route to traffic destinations [20, 12, 39, 8, 29, 31]). Similarly to low-degree networks, expanders have been shown to induce high performance in a broad spectrum of network design contexts.

Recently, the focus on either the diameter or the expansion of a network topology gave rise to two competing approaches for datacenter architecture design [46, 43, 37, 32, 8, 29, 31]. Specifically, an important line of research in datacenter design (see, e.g., [43, 37, 8, 23]) relies on (either implicitly or explicitly) utilizing graphs whose sizes are as large as possible for a given diameter and degree as datacenter network topologies[2]. A different strand of research investigates how utilizing expander graphs as datacenter network topologies can be turned into an operational reality [46, 18, 28].

[2] The authors of [37], for instance, write that "Intuitively, the best known degree-diameter topologies should support a large number of servers with high network bandwidth and low cost (small degree)... Thus, we propose the best-known degree-diameter graphs as a benchmark for comparison."

Our results show that these two approaches are, in fact, inextricably intertwined; not only do expanders exhibit low (in fact, near-optimal) diameters [26], but constructing large low-diameter datacenter networks effectively translates to constructing good expanders. Thus these two approaches to designing datacenter networks can essentially be regarded as one: the search for extremely strong expanders. Our results provide new expansion guarantees for a number of well-studied low-diameter networks, including MMS graphs [39] (proposed for the context high-performance computing and datacenters, see *Slim Fly* [8]), polarity graphs [20, 12], Canale-Gomez graphs [13], and more. See summary of the implications of our results for different graph constructions in Table 2. Our results for spectral expansion essentially match previously established results, thus generalizing and unifying prior construction-specific bounds.

Beyond the implications for network design, the study of low-diameter networks also pertains to other areas such as feedback registers [22, 35] and decoders [14].

2 Preliminaries

We provide below a brief exposition of graph expansion and the Moore Bound. We refer the reader to [26] and [41] for detailed expositions of these topics.

Let $G = (V, E)$ be an undirected graph of size $|V| = n$. G is said to be *d-regular* if each of its vertices is of degree d, and of *diameter k* if the maximum distance between any two vertices in the graph is k. d-regular graphs of diameter k are denoted throughout the paper as *(d, k)-graphs.*

The combinatorial expansion of the graph reflects an isoperimetric view and is the minimal ratio between the boundary ∂S of a set S and its cardinality. Different interpretations of ∂S give rise to different notions of expansion.

The *edge expansion* of G is

$$h_e(G) := \min_{|S| \leq \frac{n}{2}} \frac{|e(S, S^c)|}{|S|}$$

where $e(S, S^c) := \{(u, v) \in E | u \in S, v \in S^c\}$.

The *vertex expansion* of G is

$$\phi_V(G) = \min_{0<|S| \leq \frac{n}{2}} \frac{|N(S)|}{|S|}.$$

where $N(S) := \{v \in S^c | \quad \exists u \in S \text{ s.t. } (u, v) \in E\}$.[3]

We next define the algebraic (spectral) notion of expansion. Let A be the adjacency matrix of the graph. Since A is symmetric it is diagonalizable with respect to an orthonormal basis, and the corresponding eigenvalues are real, and so can be ordered as follows:

$$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n.$$

The first eigenvalue of a d-regular graph satisfies $\lambda_1 = d$ and has the all-ones vector $\mathbf{1_n}$ as the associated eigenvector. Let $\lambda(G) := \max\{|\lambda_2|, |\lambda_n|\}$. A graph G is said to be an expander if $\lambda(G)$ is bounded away from d by some constant [3]. The *algebraic expansion* (or spectral expansion) is then defined as $d - \lambda(G)$, termed the *spectral gap*[4]. The larger the gap, the better the expansion.

[3] The definitions of edge and vertex expansion admit several variants, based on either the size of the cut or the type of the boundary (see [26] for examples). While we adopt the most common of those, our results can be stated w.r.t. other variants as well.

[4] We use the two-sided notion of spectral gap throughout the paper, as oppsed to $d - \lambda_2$.

How large can a (d, k)-graph be? A straightforward upper bound is obtained by summation of the vertices according to their distance from a fixed vertex $v_0 \in V$. Let m_j denote the number of vertices at distance j from v_0. Note that $m_0 = 1$ and $m_1 = d$. As vertices at distance $j \geq 2$ must be adjacent to some vertex at distance $j - 1$, we have that $m_j \leq (d-1)m_{j-1}$. A simple induction implies that $m_j \leq d(d-1)^{j-1}$. Now since the diameter is k, all vertices have distance at most k from v_0, and hence $n \leq 1 + d + d(d-1) + d(d-1)^2 + ... + d(d-1)^{k-1}$. We denote this expression, known as the *Moore Bound* of the graph, by

$$\mu_{d,k} := 1 + d\sum_{i=0}^{k-1}(d-1)^i = \begin{cases} 2k+1 & \text{if } d = 2 \\ 1 + d \cdot \frac{(d-1)^k - 1}{d-2} & \text{if } d > 2 \end{cases}$$

3 Diameter vs. Algebraic Expansion

We establish below a relationship between the nontrivial eigenvalues of A and the distance of the graph from the Moore Bound. This relationship will enable us to prove a variety of bounds on the algebraic expansion of approximate-Moore graphs. This novel link relies on the following class of orthogonal polynomials: let $P_0(x) = 1$, $P_1(x) = x$, $P_2(x) = x^2 - d$, and for every $t > 2$ define $P_t(x)$ by the recurrence relation

$$P_t(x) = xP_{t-1}(x) - (d-1)P_{t-2}(x).$$

The significance of this class of polynomials, termed the *"Geronimus Polynomials"* [9, 44], is reflected in the main technical theorem of this section:

▶ **Theorem 1.** *Let G be (d, k)-graph of size n. Then, every nontrivial eigenvalue $\lambda < d$ of G satisfies*

$$\left|\sum_{t=0}^{k} P_t(\lambda)\right| \leq \mu_{d,k} - n$$

Before delving into the proof of Theorem 1, we discuss some of its implications. Theorem 1 can be applied to provide meaningful guarantees regarding the spectral expansion of low-diameter graphs whose sizes approach the Moore Bound. Constructing large graphs of very low diameter, e.g., $k = 2, 3$, has received much attention from both a theoretical perspective (see, e.g., [20, 12, 39]) and a practical perspective (see, e.g., [8, 29, 31]). An immediate implication of Theorem 1 is the following:

▶ **Theorem 2.** *Let G be a d-regular graph of diameter $k = 2$ and size n, then*

$$\lambda(G) \leq \frac{1 + \sqrt{1 + 4(d^2 + d - n)}}{2}.$$

Proof. Applying the Geronimus Polynomials $P_t(\lambda)$ for $0 \leq t \leq 2$ in Theorem 1 yields

$$|1 + \lambda + (\lambda^2 - d)| \leq \mu_{d,2} - n = d^2 + 1 - n.$$

The result follows from solving the quadratic inequality. ◀

This theorem immediately bounds the algebraic expansion of polarity graphs [20, 12] and MMS graphs [39] claimed in Table 2, as both of these classes of graphs have diameter 2. What about graphs of diameter $k > 2$? A more careful analysis of the Geronimus polynomials for larger values of k allows us to use Theorem 1 to prove two different expansion bounds. The first is an extremely strong expansion bound but requires the size of the graph to be additively close to the Moore bound, whereas the second allows a small multiplicative gap between the size and the Moore bound but establishes a weaker expansion guarantee.

▶ **Theorem 3.** *Let G be a (d,k)-graph of size $n \geq \mu_{d,k} - O(d^{k/2})$, for some constant $k > 0$. Then $\lambda(G) = O(\sqrt{d})$.*

Since any d-regular graph must satisfy $\lambda(G) \geq \sqrt{d}$ (see [26] for details), Theorem 3 implies that an additive approximation of $O(d^{k/2})$ of the Moore Bound implies essentially optimal spectral properties.

▶ **Theorem 4.** *Let G be a (d,k)-graph of size $n \geq (1-\varepsilon)\mu_{d,k}$, for some constant $k > 0$. Then $\lambda(G) \leq O(\varepsilon^{1/k}) \cdot d$.*

Delorme [41, 16, 17] proved the existence of an infinite series of (d,k)-graphs whose sizes arbitrarily approach the Moore Bound for diameters $k = 3$ and $k = 5$. Specifically, Delorme proved that $\liminf_{d\to\infty} \frac{n_{d,k}}{d^k} = 1$, for $k = 3, 5$, where $n_{d,k}$ is the largest possible size of a (d,k)-graph. This means that for $k = 3, 5$, for any constant $\varepsilon > 0$ there is some value d' such that for all $d \geq d'$ there is a (d,k)-graph with at least $(1-\varepsilon)\mu_{d,k}$ vertices. Hence, Theorem 4 implies that these graphs are good expanders.

Bollobás conjectured that (d,k)-graphs of size $n \geq (1-\varepsilon)d^k$ *always exist* for sufficiently large d and k [10]. Delorme's results may be perceived as supporting this conjecture. We point out that proving Bollobás' conjecture (or even extending Delorme's results to other specific values of k and d) would immediately imply, by Theorem 4, similar expansion guarantees. The remainder of this section is devoted to the proofs of Theorems 1, 3, and 4.

3.1 Bounding the nontrivial eigenvalues (proof of Theorem 1)

Our high-level approach to proving Theorem 1 is the following: We aim to bound $\lambda(G)$, the second-largest eigenvalue (in absolute value) of the adjacency matrix A. We instead consider a different matrix M, obtained by employing the *Geronimus Polynomials* as operators over A. The combinatorial properties of this class of polynomials allow us to show that $M\mathbf{1}_n = (\mu_{d,k} - n)\mathbf{1}_n$. Applying the Perron-Frobenius Theorem asserts that this eigenvalue serves as a bound over the entire spectrum of M. We then utilize the algebraic relation between both matrices: Namely, we bound A's nontrivial spectrum, using the fact that M shares the same eigenvectors as A, and that its eigenvalues may be derived from those of A via an operation of the Geronimus Polynomials. This will then imply Theorem 1.

We begin with the known solution to the recurrence, formulated via a trigonometric expression that holds for all $t > 0$ [45]:

$$P_t(2\sqrt{d-1}\cos\theta) = (d-1)^{t/2-1}\frac{(d-1)\sin((t+1)\theta) - \sin((t-1)\theta)}{\sin\theta} \tag{1}$$

One can easily check that this identity applies for $t = 1, 2$ and verify that the recurrence relation holds for $t > 2$. All roots of P_t are real and lie in the interval $[-2\sqrt{d-1}, 2\sqrt{d-1}]$ [5, 36].

Our framework applies the Geronimus Polynomials as operators over the adjacency matrix A. This method has several advantages: Algebraically, since $P_t(A)$ is a linear combination of powers of A, each eigenvector v of A is an eigenvector of $P_t(A)$ as well. Thus, the spectrum of $P_t(A)$ is given by $spec[P_t(A)] = \{P_t(\lambda) \mid \lambda \text{ is an eigenvalue of } A\}$. Viewed from a combinatorial perspective, this operation allows us to dismiss backtracking paths from consideration. By *backtracking*, we refer to paths that traverse an edge in both directions consecutively. Note that a non-backtracking path need not be simple (a nontrivial cycle is a typical example of a non-backtracking yet non-simple path). The following claim states this observation formally. The proof is straightforward and is included for completeness.

▶ **Claim 5.** *Let A be the adjacency matrix of a d-regular graph G. Then, $P_t(A)$ is the $n \times n$ matrix in which the (u, v)'th entry equals the number of non-backtracking paths of length exactly t between u and v.*

Proof. We use induction on t. Note that $P_0(A) = I$, $P_1(A) = A$ and $P_2(A) = A^2 - dI$ satisfy the claim. For the induction step, suppose that the claim holds for all Geronimus Polynomials of order strictly less than t. Consider the term $A \cdot P_{t-1}(A)$, which corresponds to paths of length t such that the first $t-1$ hops on the path are non-backtracking. Note that reducible paths in this term are those paths that can only be reduced by eliminating their last two arcs and so there must be exactly $(d-1)P_{t-2}(A)$ of them. Being the difference between those quantities, it follows that $P_t(A) = A \cdot P_{t-1}(A) - (d-1)P_{t-2}(A)$ corresponds to the non-backtracking paths. ◀

As a corollary, the entries of $P_t(A)$ are non-negative for all $t \geq 0$. In addition, as $d(d-1)^{t-1}$ is the number of non-backtracking paths of length $t > 0$ starting from every $v \in G$, this quantity equals the sum of entries in every row of $P_t(A)$. Hence, Claim 5 implies that $P_t(A)\mathbf{1}_n = d(d-1)^{t-1}\mathbf{1}_n$.

Summing over the indices $0 \leq t \leq k$, yields

$$\sum_{t=0}^{k} P_t(A)\mathbf{1}_n = \left(1 + \sum_{t=1}^{k} d(d-1)^{t-1}\right)\mathbf{1}_n = \mu_{d,k} \cdot \mathbf{1}_n. \tag{2}$$

We are now ready to prove Theorem 1:

Proof. **(of Theorem 1)** Given Claim 5, the sum of matrices $\sum_{t=0}^{k} P_t(A)$ corresponds to all non-backtracking paths of length at most k. Since G is of diameter k, this sum of matrices must consist of strictly positive entries, and can thus be represented as $\sum_{t=0}^{k} P_t(A) = J + M$, where J is the all ones matrix and M is non-negative. We now have $M\mathbf{1}_n = \left(\sum_{t=0}^{k} P_t(A) - J\right)\mathbf{1}_n = (\mu_{d,k} - n)\,\mathbf{1}_n$, where the second equality is due to (2).

Recall that A is symmetric and thus diagonalizable w.r.t. an orthogonal basis. Therefore, any eigenvector $v \neq \mathbf{1}_n$ must lie in $(span\{\mathbf{1}_n\})^{\perp}$. Since $J\mathbf{1}_n = n\mathbf{1}_n$ and $rank(J) = 1$, it follows that $Jv = 0$. Hence, $Mv = \left(\sum_{t=0}^{k} P_t(A) - J\right)v = \sum_{t=0}^{k} P_t(A)v = \sum_{t=0}^{k} P_t(\lambda)v$.

This implies in particular that

$$spec(M) = \left\{\sum_{t=0}^{k} P_t(\lambda) \mid \lambda \text{ is a nontrivial eigenvalue of } A\right\} \cup \{\mu_{d,k} - n\}.$$

We now apply the Perron-Frobenius Theorem (see [40]), which states that a non-negative matrix admits a non-negative eigenvector with a non-negative eigenvalue that is larger or equal, in absolute value, to all other eigenvalues. Now, since $\mathbf{1}_n$ is the only non-negative eigenvector of M, we conclude that $\mu_{d,k} - n$ is the leading eigenvalue of M and the claim follows. ◀

3.2 Proof of Theorem 3

Our proof of Theorem 3 utilizes a careful asymptotic estimation of the Geronimus Polynomials' coefficients. When $\lambda(G)$ is of order larger than $\sqrt{d}$, our analysis asserts that $\left|\sum_{t=0}^{k} P_t(\lambda)\right|$ must be larger then $O(d^{k/2})$ for some nontrivial eigenvalue λ of G, thus resulting in a contradiction to Theorem 1.

For our purposes, it will be beneficial to use the representation $P_t(x) = \sum_{i=0}^{t} a_{t,i}x^i$, where $a_{t,i}$ is the i'th coefficient of the t'th Geronimous Polynomial. We note the following: (i) P_t is either odd or even[5] for all $t > 0$, and the parity of P_t equals the parity of t. This can be shown either by induction using the recurrence relation, or straightforward from the solution (1); (ii) A comparison of the leading coefficients in the recurrence implies that $a_{t,t} = a_{t-1,t-1}$. Applying the boundary conditions ($a_{1,1} = a_{0,0} = 1$) yields $a_{t,t} = 1$ for all $t > 0$; (iii) Setting $\theta = \frac{\pi}{2}$ in (1) yields $a_{t,0} = d(d-1)^{t/2-1}(-1)^{t/2}$ whenever t is even.

The following easy-to-prove claim provides us with asymptotic estimates for the rest of the coefficients. Note that the $\Theta(\cdot)$ notation is hiding factors of t (we will only use this claim only where t is constant).

▶ **Claim 6.** *Let $P_t(x) = \sum_{i=0}^{t} a_{t,i}x^i$ denote the Geronimous polynomial of order t, then*

$$a_{t,i} = \begin{cases} (-1)^{\frac{t-i}{2}}\Theta\left(d^{\frac{t-i}{2}}\right) & \text{if } (t-i) \text{ is even} \\ 0 & \text{if } (t-i) \text{ is odd} \end{cases}$$

for all $0 \le i \le t$.

This immediately gives the next corollary, which is just a slightly easier to use formulation of $P_t(x)$.

▶ **Corollary 7.** *The Geronimus Polynomial of order t can be written as*

$$P_t(x) = \sum_{i=0}^{\lfloor \frac{t}{2} \rfloor} (-1)^i \cdot \Theta(d^i) \cdot x^{t-2i}.$$

We are now ready to apply this machinery. The following lemma bounds the value of these polynomials on values which are "small".

▶ **Claim 8.** *Let $\frac{1}{2} < \alpha \le 1$, and let $|\lambda| = \Theta(d^\alpha)$. Then $|P_t(\lambda)| = \Theta(d^{t\alpha})$.*

Proof. We use induction on t. For $t = 0$ we have that $P_0(\lambda) = 1 = \Theta(d^{0\alpha})$, and for $t = 1$ we have that $|P_1(\lambda)| = |\lambda| = |\Theta(d^{1\alpha})|$. Assume that the claim holds for the Geronimus Polynomials of order less than t. Using Corollary 7, we now have

$$P_t(\lambda) = \sum_{i=0}^{\lfloor \frac{t}{2} \rfloor} (-1)^i\Theta(d^i)\lambda^{t-2i} = \sum_{i=0}^{\lfloor \frac{t}{2} \rfloor} (-1)^i\Theta(d^i)\Theta(d^{\alpha(t-2i)}) = \sum_{i=0}^{\lfloor \frac{t}{2} \rfloor} (-1)^i \cdot \Theta(d^{t\alpha+i(1-2\alpha)}).$$

Whenever $\alpha > \frac{1}{2}$, the absolute value of this equals $\Theta(d^{t\alpha})$ as claimed. ◀

Proof. (of Theorem 3) Suppose that A obtains an eigenvalue $\lambda = \Theta(d^\alpha)$ for some $\alpha > \frac{1}{2}$. Then, applying Claim 8, we have:

$$\left|\sum_{t=0}^{k} P_t(\lambda)\right| = \left|\sum_{t=0}^{k} \Theta(d^{t\alpha})v\right| = \Theta(d^{k\alpha})v$$

[5] A polynomial $q(x)$ is said to be even if $q(x) = q(-x)$ and odd if $q(-x) = -q(x)$.

This expression, however, is upper bounded by $\mu_{d,k} - n$ (by Theorem 1), which is $O(d^{k/2})$ by the assumption of the Theorem 3. We thus have

$$|\Theta(d^{k\alpha})| \leq \mu_{d,k} - n \leq O(d^{k/2})$$

and this is, of course, a contradiction to the assumption $\alpha > \frac{1}{2}$. We therefore conclude that $\lambda = O(\sqrt{d})$. ◀

3.3 Proof of Theorem 4

The proof of Theorem 4 relies on some of the ideas introduced in the proof of Theorem 3. Let λ be a nontrivial eigenvalue of G. We wish to show that $|\lambda| \leq O(\varepsilon^{1/k})d$. If $|\lambda| \leq O(d^{2/3})$ then we are done. Suppose, then, that $|\lambda| \geq \omega(d^{2/3})$, and hence $d = o(|\lambda|^{3/2})$. Consider the sum $|\sum_{t=0}^{k} P_t(\lambda)|$. Corollary 7, and the discussion which showed that $a_{t,t} = 1$, imply that this sum is at least

$$\left|\sum_{t=0}^{k} P_t(\lambda)\right| \geq |\lambda^k + \lambda^{k-1}| - \sum_{i=1}^{\lfloor k/2 \rfloor} \left(\Theta(d^i)|\lambda|^{k-2i} + \Theta(d^i)|\lambda|^{k-2i-1}\right)$$
$$\geq |\lambda^k + \lambda^{k-1}| - \sum_{i=1}^{\lfloor k/2 \rfloor} \left(\Theta(|\lambda|^{k-(i/2)}) + \Theta(|\lambda|^{k-1-(i/2)})\right) \geq \Theta(|\lambda^k|),$$

where the second inequality follows from the assumption that $|\lambda| \geq \omega(d^{2/3})$.

When we plug this into Theorem 1, we get that $\Theta(|\lambda^k|) \leq \mu_{d,k} - n \leq \varepsilon\mu_{d,k}$. Since $\mu_{d,k} \leq cd^k$ for some constant c, we get that $c'|\lambda^k| \leq \varepsilon c d^k$ for some constants c and c', and hence $|\lambda| \leq \left(\frac{c}{c'}\right)^{1/k} \varepsilon^{1/k} d$, proving the theorem. ◀

4 Diameter vs. Combinatorial Expansion

We present below our results for combinatorial expansion. We first point out that applying the Cheeger inequality [26] to our bounds on spectral expansion immediately implies bounds on combinatorial expansion. Specifically, the Cheeger inequality states that $h_e(G) \geq \frac{d-\lambda_2}{2}$. When combined with Theorems 3 and 4, this yields the following bounds.

▶ **Theorem 9.** *Let G be a (d,k) graph with n vertices, for some constant $k > 0$. If $n \geq \mu_{d,k} - O(d^{k/2})$ then $h_e(G) \geq \frac{d-O(\sqrt{d})}{2}$.*

▶ **Theorem 10.** *Let G be a (d,k) graph with n vertices, for some constant $k > 0$. If $n \geq (1-\varepsilon)\mu_{d,k}$ then $h_e(G) \geq \frac{(1-O(\varepsilon^{1/k}))d}{2}$.*

Observe that, since clearly $d/2$ is an upper bound on $h_e(G)$, both of these bounds imply very high expansion guarantees when n is very close to the Moore Bound. However, when this is not so, e.g., when $n = \mu_{d,k}/k$, neither bound yields nontrivial expansion guarantees.

To provide stronger expansion guarantees for graphs that do not come very close (additively/ multiplicatively) to the Moore Bound, we analyze combinatorial expansion directly. We next present our bounds for edge and vertex expansion in undirected and directed graphs. We discuss the implications of these expansion bounds for known (d,k)-graph constructions in Table 2.

Our main result of this section is the following:

▶ **Theorem 11.** *Let $G = (V, E)$ be a d-regular graph of size n and diameter k. If $n = \alpha \cdot \mu_{d,k}$, then*

$$h_e(G) \geq \frac{\alpha d}{2k} \cdot \left(1 - \frac{1}{(d-1)^k}\right) \quad \textit{and} \quad \phi_V(G) \geq \frac{\alpha}{2(k-1)+\alpha}.$$

Our proof of Theorem 11 utilizes a counting argument. As the graph has diameter k, each pair of vertices on opposite sides of a cut must be connected via a path of length at most k that traverses the boundary. However, there is an upper bound, induced by the degree and diameter of the graph, on the number of such paths that traverse a given edge/vertex. A careful examination of the implications of these two limitations provides us with a lower bound on the size of the boundary.

Proof. (first part of Theorem 11) Let (S, S^c) be a cut in the graph, and let $|S| = s \leq \frac{n}{2}$. As the diameter equals k, every pair of vertices that lie on both sides of the cut must be connected via a path of length at most k. We thus have $s(n-s)$ such paths, each of which passes through some edge in the cut.

How many paths of length at most k include a given edge $e \in E$? As G is d-regular, there are at most $(d-1)^{l-1}$ paths of length l for which e is in the i'th position in the path. It follows that no more than $l \cdot (d-1)^{l-1}$ paths of length l use a specific edge, hence the number of paths of length at most k that utilize a fixed edge is upper bounded by

$$f_{d-1}(k) = \sum_{l=1}^{k} l \cdot (d-1)^{l-1}.$$

Let us find a simpler formulation of $f_{d-1}(k)$. Integrating yields

$$F_{d-1}(k) = \sum_{l=1}^{k} (d-1)^l = \frac{(d-1)^{k+1}-1}{(d-1)-1}.$$

Differentiating brings us back to

$$\begin{aligned} f_{d-1}(k) &= \frac{(k+1)(d-1)^k(d-2) - [(d-1)^{k+1}-1]}{(d-2)^2} \\ &\leq \frac{(k+1)(d-1)^k(d-2) - (d-1)^k(d-2)}{(d-2)^2} \\ &= \frac{k(d-1)^k}{(d-2)} \end{aligned}$$

Now, $s(n-s)$ paths use the cut, and every edge in the cut can be a part of at most $f_{d-1}(k)$ paths. It follows that $|e(S, S^c)| \geq \frac{s(n-s)}{f_{d-1}(k)}$ for every cut (S, S^c) in G. Hence, the cut that realizes $h_e(G)$ satisfies

$$\begin{aligned} h_e(G) &= \frac{|e(S, S^c)|}{|S|} \geq \frac{s(n-s)(d-2)}{s \cdot k(d-1)^k} \geq \frac{n}{2} \cdot \frac{(d-2)}{k(d-1)^k} \\ &= \frac{\alpha d \cdot ((d-1)^k - 1)}{2(d-2)} \cdot \frac{(d-2)}{k(d-1)^k} \\ &= \frac{\alpha d}{2k} \cdot \left(1 - \frac{1}{(d-1)^k}\right). \end{aligned}$$

◀

4.1 Directed graphs.

We consider directed graphs next. We begin by introducing the relevant terminology and notation. We say that a directed graph (a.k.a. digraph) $G = (V, E)$ is d-regular if both the out-degree and the in-degree of each vertex equals d. A cut in a digraph $e(S, S^c) = \{(u, v) \in E | u \in S, v \in S^c\}$ is asymmetric, and consists of all edges directed from S to S^c. The diameter is still defined as the maximal distance between two vertices, and the corresponding Moore Bound is only slightly different (as there are potentially d^i vertices of distance i from a given vertex): $\tilde{\mu}_{d,k} = \sum_{i=0}^{k} d^i = \frac{d^{k+1}-1}{d-1}$.

The following result is the directed analogue of Theorem 11:

▶ **Theorem 12.** *Let G be a d-regular, k-diameter **directed** graph of size $n = \alpha \cdot \tilde{\mu}_{d,k}$, then*

$$h(G) \geq \frac{\alpha}{2k}\left(d - \frac{1}{d^k}\right) \quad \text{and} \quad \phi_V(G) \geq \frac{\alpha \cdot d}{2(d+1)(k-1) + \alpha \cdot d}.$$

Much research on constructing low-diameter graphs focuses on diameters 2 and 3 (see, e.g., [41, 20, 39]). Graphs of very low diameter are particularly important from a practical perspective [8, 29, 31]. The following theorems improve upon our results for the edge expansion and vertex expansion of (d, k)-graphs.

▶ **Theorem 13.** *Let $G = (V, E)$ be an undirected $(d, 2)$-graph of size $n = \alpha \cdot d^2$. Then*

$$h_e(G) \geq \frac{2d + 1 - \sqrt{4(1-\alpha)d^2 + 4d + 1}}{4}.$$

▶ **Theorem 14.** *Let $G = (V, E)$ be an undirected $(d, 2)$-graph of size $n = \alpha \cdot d^2$. Then $\phi_V(G) \geq \frac{2\alpha}{2\alpha+1}$.*

We can extend our analysis to graphs of diameter 3, yielding the following theorem.

▶ **Theorem 15.** *Let $G = (V, E)$ be a $(d, 3)$-graph of size $n = \alpha \cdot d^3$, then $\phi_V(G) \geq \frac{\alpha}{\alpha+1}$.*

5 Conclusion and Open Questions

We revisited the classical question of relating the expansion and the diameter of graphs and showed that not only do good expanders exhibit low diameter but the converse is also, in some sense, true. We also discussed the implications of our results for constructions from the rich body literature on low-diameter graphs. We leave the reader with many interesting open questions, including: (1) **Tightening the gaps.** An obvious open question is improving upon our lower bounds and establishing upper bounds on the expansion of fixed-diameter graph constructions. (2) **Benchmarking against the optimal (largest possible) (d, k)-graph.** We used the Moore Bound as a benchmark. Another approach would be to compare against the size of the largest possible (d, k)-graph. (3) **Geronimus Polynomials vs. Hashimoto's non-backtracking operator.** The operation of the Geronimus Polynomials over the adjacency matrix of the graph offers a new perspective on its non-backtracking paths (as established in Lemma 5). This suggests a non-trivial relation between these polynomials and Hashimoto's non-backtracking operator, which we believe is of independent interest and may find wider applicability.

References

1 Noga Alon, Itai Benjamini, Eyal Lubetzky, and Sasha Sodin. Non-backtracking random walks mix faster. *Communications in Contemporary Mathematics*, 9(04):585–603, 2007.

2 Baba Arimilli, Ravi Arimilli, Vicente Chung, Scott Clark, Wolfgang Denzel, Ben Drerup, Torsten Hoefler, Jody Joyner, Jerry Lewis, Jian Li, et al. The percs high-performance interconnect. In *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*, pages 75–82. IEEE, 2010.

3 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach.* Cambridge University Press, 2009.

4 Eiichi Bannai and Tatsuro Ito. Regular graphs with excess one. *Discrete Mathematics*, 37(2):147–158, 1981.

5 Yair Bartal, Nathan Linial, Manor Mendel, and Assaf Naor. On metric ramsey-type phenomena. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 463–472. ACM, 2003.

6 J-C Bermond, Nathalie Homobono, and Claudine Peyrat. Large fault-tolerant interconnection networks. *Graphs and Combinatorics*, 5(1):107–123, 1989.

7 Jean-Claude Bermond. De bruijn and kautz networks: a competitor for the hypercube? *Hypercube and distributed computers*, pages 279–293, 1989.

8 Maciej Besta and Torsten Hoefler. Slim fly: A cost effective low-diameter network topology. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 348–359. IEEE Press, 2014.

9 Norman Biggs. *Algebraic graph theory.* Cambridge university press, 1993.

10 B Bollobás. *Extremal Graph Theory.* Dover Publications, 1978.

11 C Bordenave. A new proof of friedman's second eigenvalue theorem and its extension to random lifts. *Preprint, available at* `https://arxiv.org/abs/1502.04482`, 2015.

12 William G Brown. On graphs that do not contain a thomsen graph. *Canad. Math. Bull*, 9(2):1–2, 1966.

13 Eduardo A Canale and José Gómez. Asymptotically large (δ, d)-graphs. *Discrete applied mathematics*, 152(1):89–108, 2005.

14 Oliver Collins, Sam Dolinar, Robert McEliece, and Fabrizio Pollara. A vlsi decomposition of the de bruijn graph. *Journal of the ACM (JACM)*, 39(4):931–948, 1992.

15 DG De Bruijn. A combinatorial problem, koninklijke nederlandsche academie van wetenschappen et amsterdam. *Proceedings Qt the Section gt Sciences*, 49:27, 1946.

16 Charles Delorme. Grands graphes de degré et diametre donnés. *European Journal of Combinatorics*, 6(4):291–302, 1985.

17 Charles Delorme. Large bipartite graphs with given degree and diameter. *Journal of graph theory*, 9(3):325–334, 1985.

18 Michael Dinitz, Michael Schapira, and Asaf Valadarsky. Explicit expanding expanders. *Algorithmica*, pages 1–21, 2016. `doi:10.1007/s00453-016-0269-x`.

19 Bernard Elspas, William H Kautz, and James Turner. Theory of cellular logic networks and machines. Technical report, DTIC Document, 1968.

20 Paul Erdos, Alfréd Rényi, and VT Sós. On a problem in the theory of graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 7:215–235, 1962.

21 A-H Esfahanian and S. Louis Hakimi. Fault-tolerant routing in de bruijn comrnunication networks. *IEEE Transactions on Computers*, 100(9):777–788, 1985.

22 Harold Fredricksen. A survey of full length nonlinear shift register cycle algorithms. *SIAM review*, 24(2):195–221, 1982.

23 D. Guo, T. Chen, D. Li, M. Li, Y. Liu, and G. Chen. Expandable and cost-effective network structures for data centers using dual-port servers. *IEEE Transactions on Computers*, 62(7):1303–1317, July 2013. `doi:10.1109/TC.2012.90`.

24 Ki-ichiro Hashimoto. Zeta functions of finite graphs and representations of p-adic groups. *Automorphic forms and geometry of arithmetic varieties.*, pages 211–280, 1989.

25 Alan J Hoffman and Robert R Singleton. On moore graphs with diameters 2 and 3. *IBM Journal of Research and Development*, 4(5):497–504, 1960.

26 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.

27 Brian Karrer, Mark EJ Newman, and Lenka Zdeborová. Percolation on sparse networks. *Physical review letters*, 113(20):208702, 2014.

28 Simon Kassing, Asaf Valadarsky, Gal Shahaf, Michael Schapira, and Ankit Singla. Beyond fat-trees without antennae, mirrors, and disco-balls. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 281–294. ACM, 2017.

29 John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. In *ACM SIGARCH Computer Architecture News*, volume 36, pages 77–88. IEEE Computer Society, 2008.

30 John Kim, William J Dally, and Dennis Abts. Flattened butterfly: a cost-efficient topology for high-radix networks. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 126–137. ACM, 2007.

31 John Kim, William J Dally, Steve Scott, and Dennis Abts. Cost-efficient dragonfly topology for large-scale systems. In *Optical Fiber Communication Conference*, page OTuI2. Optical Society of America, 2009.

32 Michihiro Koibuchi, Hiroki Matsutani, Hideharu Amano, D Frank Hsu, and Henri Casanova. A case for random shortcut topologies for hpc interconnects. In *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, pages 177–188. IEEE, 2012.

33 Florent Krzakala, Cristopher Moore, Elchanan Mossel, Joe Neeman, Allan Sly, Lenka Zdeborová, and Pan Zhang. Spectral redemption in clustering sparse networks. *Proceedings of the National Academy of Sciences*, 110(52):20935–20940, 2013.

34 F Thomson Leighton. *Introduction to parallel algorithms and architectures: Arrays · trees · hypercubes*. Elsevier, 2014.

35 Abraham Lempel. On a homomorphism of the de bruijn graph and its applications to the design of feedback shift registers. *IEEE Transactions on Computers*, 100(12):1204–1209, 1970.

36 Nathan Linial, Avner Magen, and Assaf Naor. Girth and euclidean distortion. *Geometric & Functional Analysis GAFA*, 12(2):380–394, 2002.

37 Ankit Singla Chi-Yao Hong Lucian and Popa Brighten Godfrey. Jellyfish: Networking data centers randomly. *CoRR*, abs/1110.1687, 2011. URL: `http://arxiv.org/abs/1110.1687`.

38 Travis Martin, Xiao Zhang, and MEJ Newman. Localization and centrality in networks. *Physical Review E*, 90(5):052808, 2014.

39 Brendan D McKay, Mirka Miller, and Jozef Širáň. A note on large graphs of diameter two and given maximum degree. *Journal of Combinatorial Theory, Series B*, 74(1):110–118, 1998.

40 Carl D Meyer. *Matrix analysis and applied linear algebra*, volume 2. Siam, 2000.

41 Mirka Miller and Jozef Širán. Moore graphs and beyond: A survey of the degree/diameter problem. *Electronic Journal of Combinatorics*, 61:1–63, 2005.

42 Dhiraj K Pradhan. Fault-tolerant multiprocessor link and bus network architectures. *IEEE Trans. Comput.;(United States)*, 100, 1985.

43 Ankit Singla, P Brighten Godfrey, and Alexandra Kolla. High throughput data center topology design. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 29–41, 2014.

44 Patrick Solé. The second eigenvalue of regular graphs of given girth. *Journal of Combinatorial Theory, Series B*, 56(2):239–249, 1992.

45 Gabor Szeg. *Orthogonal polynomials*, volume 23. American Mathematical Soc., 1939.

46 Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. Xpander: Towards optimal-performance datacenters. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, pages 205–219, New York, NY, USA, 2016. ACM. doi:10.1145/2999572.2999580.

Improved Dynamic Graph Coloring

Shay Solomon
IBM Research, TJ Watson Research Center, Yorktown Heights, NY USA

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

Abstract

This paper studies the fundamental problem of graph coloring in fully dynamic graphs. Since the problem of computing an optimal coloring, or even approximating it to within $n^{1-\epsilon}$ for any $\epsilon > 0$, is NP-hard in *static graphs*, there is no hope to achieve any meaningful *computational* results for *general graphs* in the dynamic setting. It is therefore only natural to consider the *combinatorial aspects* of dynamic coloring, or alternatively, study restricted families of graphs.

Towards understanding the combinatorial aspects of this problem, one may assume a black-box access to a static algorithm for C-coloring any subgraph of the dynamic graph, and investigate the trade-off between the number of colors and the number of *recolorings* per update step. Optimizing the number of recolorings, sometimes referred to as the *recourse* bound, is important for various practical applications. In WADS'17, Barba et al. devised two complementary algorithms: For any $\beta > 0$, the first (respectively, second) maintains an $O(C\beta n^{1/\beta})$ (resp., $O(C\beta)$)-coloring while recoloring $O(\beta)$ (resp., $O(\beta n^{1/\beta})$) vertices per update. Barba et al. also showed that the second trade-off appears to exhibit the right behavior, at least for $\beta = O(1)$: Any algorithm that maintains a c-coloring of an n-vertex *dynamic forest* must recolor $\Omega(n^{\frac{2}{c(c-1)}})$ vertices per update, for any constant $c \geq 2$. Our contribution is two-fold:

- We devise a new algorithm for general graphs that improves significantly upon the first trade-off in a wide range of parameters: For any $\beta > 0$, we get a $\tilde{O}(\frac{C}{\beta}\log^2 n)$-coloring with $O(\beta)$ recolorings per update, where the $\tilde{O}$ notation supresses polyloglog(n) factors. In particular, for $\beta = O(1)$ we get constant recolorings with polylog(n) colors; not only is this an exponential improvement over the previous bound, but it also unveils a rather surprising phenomenon: The trade-off between the number of colors and recolorings is highly non-symmetric.
- For uniformly sparse graphs, we use low out-degree orientations to strengthen the above result by bounding the update time of the algorithm rather than the number of recolorings. Then, we further improve this result by introducing a new data structure that refines bounded out-degree edge orientations and is of independent interest.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Theory of computation → Graph algorithms analysis, Theory of computation → Dynamic graph algorithms

Keywords and phrases coloring, dynamic graph algorithms, graph arboricity, edge orientations

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.72

Acknowledgements The authors thank Krzysztof Onak, Baruch Schieber and Virginia Vassilevska Williams for fruitful discussions.

1 Introduction

1.1 Background

Graph coloring is one of the most fundamental and well studied problems in computer science, having found countless applications over the years, ranging from scheduling and computational vision to biology and chemistry. A *proper C-coloring* of a graph $G = (V, E)$,

© Shay Solomon and Nicole Wein;
licensed under Creative Commons License CC-BY

for a positive integer C, assigns a color in $\{1, \ldots, C\}$ to every vertex, so that no two adjacent vertices are assigned the same color. The *chromatic number* of the graph is the smallest integer C for which a proper C-coloring exists. (We shall write "coloring" as a shortcut for "proper coloring", unless otherwise specified.)

This paper studies the problem of graph coloring in fully dynamic graphs subject to edge updates. A *dynamic graph* is a graph sequence $\mathcal{G} = (G_0, G_1, \ldots, G_M)$ on a fixed vertex set V, where the initial graph is $G_0 = (V, \emptyset)$ and each graph $G_i = (V, E_i)$ is obtained from the previous graph G_{i-1} in the sequence by either adding or deleting a single edge.

We investigate general graphs as well as *uniformly sparse* graphs. The "uniform density" of the graph is captured by its *arboricity*: a graph $G = (V, E)$ has *arboricity* α if $\alpha = \max_{U \subseteq V} \left\lceil \frac{|E(U)|}{|U|-1} \right\rceil$, where $E(U) = \{(u, v) \in E \mid u, v \in U\}$. That is, the arboricity is close to the maximum *density* $|E(U)|/|U|$ over all induced subgraphs of G. The class of constant arboricity graphs, which contains planar graphs, bounded tree-width graphs, and in general all minor-free graphs, as well as some classes of "real-world" graphs, has been subject to extensive research in the dynamic algorithms literature [10, 11, 50, 56, 46, 47, 39, 25, 14]. A dynamic graph of *arboricity* α is a dynamic graph such that all graphs G_i have arboricity bounded by α.

It is NP-hard to approximate the chromatic number of an n-vertex graph to within a factor of $n^{1-\epsilon}$ for any constant $\epsilon > 0$, let alone to compute the corresponding coloring [60, 33]. Consequently, there is no hope to achieve any meaningful *computational* results for *general graphs* in the dynamic setting. It is perhaps for that reason that the literature on dynamic graph coloring is sparse (see Section 1.1.1). Nevertheless, as discussed next, one may view the area of dynamic graph algorithms as lying within the wider area of *local algorithms*, in which there has been tremendous success in the context of graph coloring.

When dealing with networks of large scale, it is important to devise algorithms that are intrinsically *local*. Roughly speaking, a local algorithm restricts its execution to a small part of the network, yet is still able to solve a global task over the entire network. There is a long line of work on local algorithms for graph coloring and related problems from various perspectives. For example, seminal papers on distributed graph coloring [16, 26, 40, 4, 41, 42] laid the foundation for the area of symmetry breaking problems, which remains the subject of ongoing intensive research. Refer to the book of Barenboim and Elkin [7] for a detailed account on this topic. Additionally, graph coloring is well-studied in the areas of property testing [27, 17] and local computation algorithms [52, 24].

1.1.1 Dynamic graph coloring

In light of the computational intractability of graph coloring, previous work on dynamic graph coloring is devoted mostly to heuristics and experimental results [43, 51, 59, 29, 28, 48, 53]. From the theoretical standpoint, it is natural to consider the *combinatorial aspects* of dynamic coloring or to study restricted families of graphs; to the best of our knowledge, the only work on this pioneering front is that of Barba et al. from WADS'17 [5] and Bhattacharya et al. from SODA'18 [12]. Additionally, Parter, Peleg, and Solomon [49] studied this problem in the dynamic distributed setting, and Barenboim and Maimon [8] studied the related problem of dynamic *edge coloring*. (Our work focuses on amortized time bounds; we henceforth do not distinguish between amortized and worst-case time bounds, unless explicitly specified.)

Barba et al. [5] studied the combinatorial aspects of dynamic coloring in general graphs. They assumed that at all times the graph can be C-colored and further assumed black-box access to a static algorithm for C-coloring any subgraph of the current graph. They investigated the trade-off between the number of colors and the number of *recolorings* (i.e., the number of vertices that change their color) per update step. Optimizing the number of recolorings, sometimes referred to as the *recourse* bound, is important for various practical applications They devised two complementary algorithms: for any $\beta > 0$, the first (respectively, second) maintains an $O(C\beta n^{1/\beta})$ (resp., $O(C\beta)$)-coloring while recoloring $O(\beta)$ (resp., $O(\beta n^{1/\beta})$) vertices per update step. While these trade-offs coincide at $\beta = \log n$, each providing $O(C \log n)$-coloring with $O(\log n)$ recolorings per update, any slight improvement on one of these parameters triggers a significant blowup to the other. In particular, the extreme point $\beta = O(1)$ on the first and second trade-off curves yields a polynomial number of colors and recolorings, respectively. Barba et al. [5] also showed that the second trade-off exhibits the right behavior, at least for $\beta = O(1)$: Any algorithm that maintains a c-coloring of an n-vertex *dynamic forest* must recolor $\Omega(n^{\frac{2}{c(c-1)}})$ vertices per update, for any constant $c \geq 2$. The following question was left open.

▶ **Question 1.1.** *Does the first trade-off of [5] exhibit the right behavior, and in particular, does a constant number of recolorings require a polynomial number of colors?*

Bhattacharya et al. [12] studied the problem of dynamically coloring bounded degree graphs. For graphs of maximum degree Δ they presented a randomized (respectively deterministic) algorithm for maintaining a $(\Delta+1)$ (resp., $\Delta(1+o(1))$-coloring with amortized expected $O(\log \Delta)$ (resp., polylog(Δ)) update time. These results provide meaningful bounds only when *all vertices* have bounded degree. The following question naturally arises.

▶ **Question 1.2.** *Can we get meaningful results for the more general class of bounded arboricity graphs?*

Question 1.2 is especially intriguing because, as shown in [5], dynamic forests (which have arboricity 1) appear to provide a hard instance for dynamic graph coloring.

Parter, Peleg, and Solomon [49] studied Question 1.2 in dynamic distributed networks: They showed that for graphs of arboricity α an $O(\alpha \cdot \log^* n)$-coloring can be maintained with $O(\log^* n)$ update time. The update time in this context, however, bounds the number of *communication rounds* per update, while the number of recolorings done (and number of messages sent) per update is polynomial in n, even for forests.

1.2 Our results

We use $\tilde{O}$ notation throughout to suppress polyloglog factors.

1.2.1 General graphs

The following theorem summarizes our main result for general graphs.

▶ **Theorem 1.3.** *For any n-vertex dynamic graph that can be C-colored at all times, there is a fully dynamic deterministic algorithm for maintaining an $O(\frac{C}{\beta}\log^3 n)$-coloring with $O(\beta)$ (amortized) recolorings per update step, for any $\beta > 0$. Using randomization (against an oblivious adversary), the number of colors can be reduced by a factor of $\tilde{O}(\log n)$ while achieving an expected bound of $O(\beta)$ recolorings.*

Theorem 1.3 with $\beta = O(1)$ yields $O(1)$ recolorings with polylog(n) colors, thus answering Question 1.1 in the negative. Not only is this result an exponential improvement over the previous bound of [5], but it also unveils a rather surprising phenomenon: The trade-off between the number of colors and recolorings is highly non-symmetric.

We also note that the number of recolorings can be de-amortized. The details are omitted due to space constraints.

A runtime bound. Assuming black-box access to two efficient coloring algorithms we can bound the runtime of the algorithm from Theorem 1.3.

Black-box static algorithm. Let $A_{\mathcal{G},C}$ be a static algorithm that takes as input a graph G from a graph class $\mathcal{G}$ and a subset S of vertices in G, and computes the induced graph $G[S]$ and a C-coloring of $G[S]$ in time $T(|S|)$.

Black-box dynamic algorithm. Let A' be a fully dynamic algorithm that colors graphs of maximum degree Δ using $O(\Delta)$ colors. Such algorithms exist: there is a randomized algorithm with $O(1)$ expected amortized update time and a deterministic algorithm with $O(\text{polylog}(\Delta))$ amortized update time [12]. Let $T'(\Delta, n) \leq \text{polylog}(\Delta)$ be the runtime of an optimal deterministic algorithm for this problem. We state our results in terms of $T'(\Delta, n)$ to emphasize that any improvement over the deterministic algorithm of [12] would yield an improvement to the runtime of our algorithm.

▶ **Theorem 1.4.** *The randomized algorithm from Theorem 1.3 has expected amortized update time $O\left(\frac{\beta}{n\log n}\sum_{i=0}^{\log n} 2^i T(n/2^i)\right)$ and the deterministic algorithm from Theorem 1.3 has the same amortized update time with an additional additive factor of $T'(\frac{\log^2 n}{\beta}, n)$.*

▶ Remark. The randomized black-box dynamic algorithm of [12] that we apply in Theorem 1.4 is actually a simple observation (referred to as a "warm-up result" in [12]) which gives a 2Δ-coloring with $O(1)$ expected update time. The main result of [12], however, is an algorithm to bound the number of colors by only $\Delta + 1$ (or slightly more).

1.2.2 Uniformly sparse graphs

We answer Question 1.2 in the positive by showing that by applying the algorithms from Theorem 1.4 to arboricity α graphs we can obtain a bound on the update time rather than only the number of recolorings.

▶ **Theorem 1.5.** *There is a fully dynamic deterministic algorithm for graphs of arboricity α that maintains an $O((\frac{\alpha}{\beta})^2 \log^4 n)$-coloring in amortized $T'(\frac{\alpha\log^3 n}{\beta^2}, n) + O(\beta)$ time per update for any $\beta > 0$. Using randomization (against an oblivious adversary), the number of colors can be reduced by a factor of $\tilde{O}(\log n)$ and the expected amortized update time becomes $O(\beta)$.*

Furthermore, we improve over this result when $\beta = o(\sqrt{\log n})$ by designing an algorithm that specifically exploits the structure of arboricity α graphs.

▶ **Theorem 1.6.** *There is a fully dynamic deterministic algorithm for graphs of arboricity α that maintains an $O(\alpha\log^2 n)$-coloring in amortized $\tilde{O}(1)$ time.*

The proof of Theorem 1.6 relies on a new *layered data structure* (LDS) for bounded arboricity graphs that we expect will be more widely applicable. See Section 1.3 for more details about the LDS and the related notion of bounded out-degree edge orientations.

▶ **Definition 1.7.** Given a dynamic graph G of arboricity α, a *layered data structure (LDS)* with parameters k and Δ is a partition of the vertices into k layers $L_1, \ldots, L_k$ so that all vertices v have at most Δ neighbors in layers equal to or higher than the layer containing v.

▶ **Theorem 1.8.** *Let A'' be an algorithm for arboricity α graphs that maintains an orientation of the edges with out-degree at most D that performs amortized $F(n)$ flips per update. Then there is an algorithm to maintain an LDS for a fully dynamic graph of arboricity α with $k = O(\log n)$ and $\Delta = O(D + \alpha \log n)$ in amortized deterministic time $O(F(n))$.*

1.3 Technical overview

1.3.1 Low out-degree dynamic edge orientations

All of our results are, in different ways, intimately related to the dynamic edge orientation problem for arboricity α graphs, where the goal is to dynamically maintain a low out-degree orientation of the edges in a graph (an orientation with out-degree α always exists [45]). Our algorithm for general graphs (outlined in Section 1.3.2) is inspired by an algorithm for the dynamic edge orientation problem. Our algorithm for bounded arboricity graphs from Theorem 1.5 uses a dynamic edge orientation algorithm as a black-box. Our algorithm for bounded arboricity graphs from Theorem 1.6 uses a dynamic edge orientation to define a potential function useful in the runtime analysis (outlined in Section 3.1).

Brodal and Fagerberg [14] initiated the study of the dynamic edge orientation problem and gave an algorithm that maintains an $O(\alpha)$ out-degree orientation in amortized $O(\alpha + \log n)$ time. To analyze this algorithm, they reduced the "online" setting, where we have no knowledge of the future, to the "offline" settings, where we know the entire sequence of edge updates in advance. Thus, in the the subsequent results, it sufficed to consider only the offline setting. Kowalik [36] used an elegant argument to derive a result complementary to [14]: one can maintain an $O(\alpha \log n)$ out-degree orientation in amortized $O(1)$ time. He, Tang, and Zeh [30] completed the picture with a trade-off bound: for all $\beta \geq 1$, one can maintain an $O(\beta\alpha)$ out-degree orientation in amortized $O(\frac{\log n}{\beta})$ time. The worst-case update time of this problem has also been studied by Kopelowitz et al. [34] and Berglin and Brodal [9].

Dynamic bounded out-degree orientations are a key ingredient in a number of dynamic algorithms for graphs of bounded arboricity [37, 35, 10, 11, 46, 47, 25, 22], as well as in dynamic algorithms for general graphs [55, 10, 11, 13].

1.3.2 Overview of algorithm for general graphs

We apply two black-box coloring algorithms defined in Section 1.2.1, one static and one dynamic. For each vetex v, if it is assigned color c_1 by the static algorithm and color c_2 by the dynamic algorithm, its true color is defined by the pair (c_1, c_2).

Periodically, we run the static algorithm on a carefully chosen induced subgraph of the current graph. To select these subgraphs, we keep track of the *recent degree* of each vertex v: the number of edges incident to v that were inserted since the last time v was included as input to an instance of the static algorithm. Then, we choose the vertices of highest recent degree as input to the static algorithm, thus setting the recent degree of these vertices to zero. By repeatedly setting the recent degree of the highest recent degree vertices to zero, we obtain a bound on the maximum recent degree in the graph. Then we apply the dynamic algorithm for bounded degree graphs on only the edges that contribute to recent degrees.

We can further reduce the maximum recent degree in the graph by employing randomization: In addition to the vertices already chosen to participate in the static algorithm, we randomly select some vertices incident to newly inserted edges.

To obtain an upper bound on the maximum recent degree at all times, we model the changes in recent degree by an online 2-player balls and bins game. The game was first introduced in 1988 [38, 19] and has found a number of applications in the dynamic algorithms literature for obtaining worst-case guarantees [20, 15, 1, 57, 2, 44, 57, 58, 9, 21, 32]. To the best of our knowledge, our techniques are the first to demonstrate improved amortized guarantees using the game. We anticipate that this game will find additional applications in amortized algorithms as well as in translating offline strategies to online strategies.

The main technical content that remains are the details of each instance of the static algorithm: we have not specified when to run each instance, the precise subgraph to input, and which palette of colors to draw from. Understanding these details illuminates the key insight that allows us to improve the number of colors from the polynomial bound in [5] to polylogarithmic. We hierarchically bipartition the update sequence into $\log_2 n$ levels of nested time intervals and at the end of each interval, we apply the static algorithm. We use a separate palette of colors for each level of intervals but for subsequent instances of the static algorithm on the same level we use the same palette. To avoid color conflicts caused by this reuse of colors, we ensure that when a vertex participates in an instance of the static algorithm at the end of an interval I on a level L, it also participates in the instance of the static algorithm at the end of every superinterval of I; thus it is recolored once for each superinterval. This recoloring frees the level L color palette for future instances of the static algorithm at level L. In summary, the hierarchical partition of the update sequence into levels provides the structure that allows us to reuse colors without creating color conflicts.

This partition of the update sequence is inspired by the offline algorithm of [30] for the dynamic edge orientation problem. Adapting their ideas to our setting requires overcoming two main hurdles: a) transitioning from graphs of bounded arboricity to general graphs, and b) transitioning from the offline setting to the online setting.

1.3.3 Overview of algorithm for low arboricity graphs

The proof of Theorem 1.5 is based on the following observation: the black-box static algorithm used in Theorem 1.3 can be made efficient if $\mathcal{G}$ is the class of arboricity α graphs and we have access to a low out-degree orientation of the graph.

The bulk of the proof of Theorem 1.6 concerns the LDS (defined in Section 1.2.2). The definition of the LDS is inspired by the following property of arboricity α graphs: there exists an ordering of the vertices $v_1, \ldots, v_n$ such that every vertex has at most 2α neighbors that appear after it in the ordering [3]. Given such an ordering, consider the procedure of iteratively removing the vertices from the graph in order (or adding the vertices to the graph in reverse order) so that when each vertex is removed (or added) its degree to the current graph is only 2α. This procedure has been a key ingredient in algorithms in a variety of settings including distributed algorithms [6], parallel algorithms [3], property testing [23], and social network analysis [31, 54, 18]. We are the first to devise a data structure that dynamically maintains (an approximate version of) this ordering.

The LDS is useful for maintaining a proper coloring of a graph because the graph induced by each layer of vertices has low degree. Thus, we can apply a dynamic algorithm for graphs of bounded maximum degree on the graph induced by each individual layer. Then, because there are not too many layers in total, we can use a disjoint palette of colors for each layer.

On the other hand, simply using a low out-degree orientation of the edges does not seem to suffice for solving dynamic coloring. In general, one shortfall of a low out-degree orientation is that it is an inherently *local* data structure; each vertex only keeps track of information about its immediate neighborhood. In contrast, an LDS maintains a *global*

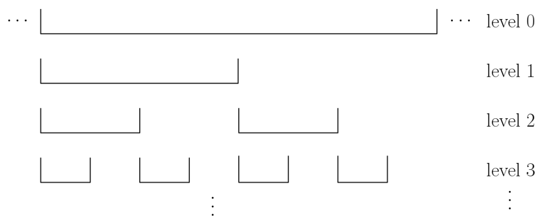


Figure 1 The set of interals.

partition of the vertices into layers. Furthermore, the LDS is designed to store strictly more information than a bounded out-degree edge orientation; by orienting all edges in an LDS from lower to higher layers, we get a bounded out-degree edge orientation. We anticipate that the LDS could be useful for solving more dynamic problems for which a bounded out-degree edge orientation does not appear to suffice.

2 Algorithm for general graphs

In this section we prove Theorem 1.3. We omit the proof of Theorem 1.4.

▶ **Theorem 2.1** (Restatement of Theorem 1.3). *There is a fully dynamic deterministic algorithm for maintaining an $O(\frac{C}{\beta}\log^3 n)$-coloring with $O(\beta)$ (amortized) recolorings per update step, for any $\beta > 0$. Using randomization (against an oblivious adversary), the number of colors can be reduced to $O(\frac{C}{\beta}\log^2 n(\log\log n + \log\beta))$ while achieving an expected bound of $O(\beta)$ recolorings.*

The algorithm is as follows. At all times, each vertex v is assigned a color c_1 by the black-box static algorithm (from the last time v was input to an instance of the static algorithm) and a color c_2 by the black-box dynamic algorithm. The true color of each v defined by the pair (c_1, c_2), so the total number of colors is the product of the number of colors used in each black-box algorithm. As mentioned in the algorithm overview (Section 1.3.2), we define a hierarchical partition of the update sequence to specify the instances of the static algorithm. First, we describe this partition, then we describe how to apply the static algorithm, and then we describe how to apply the dynamic algorithm.

2.1 Partition of update sequence

We partition the update sequence (without knowing its contents) into a set of intervals as follows. An interval is said to be of *length* ℓ if it contains ℓ update steps. We partition the entire update sequence into intervals of length $n\ell$ for some parameter ℓ (which we will later set to $\frac{\log n}{\beta}$). We say that this set of intervals is on *level* 0. Next, for each $i = 1, \ldots, \log_2 n$, the level-i intervals are obtained from the $i-1$-level intervals by splitting each $i-1$ interval in two subintervals of equal length. Note that the intervals on level $\log n$ are of length ℓ and in general the intervals on level i are of length $n\ell/2^i$.

It will be easier to work with these intervals if no two have the same ending point. So, for every pair of intervals with the same endpoint, we remove the interval on the higher numbered level. The resulting set of intervals, shown in Figure 1 is the set of intervals that we work with in the algorithm.

2.2 Applying the black-box static algorithm

At the end of each interval, we apply the black-box static algorithm. For each interval I, let $\mathcal{A}_I$ be the instance of the black-box algorithm that is executed at the end of interval I. If I is an interval on level i, we say that $\mathcal{A}_I$ is on level i. For each level, we use a separate palette of C colors, and all instances of the algorithm on the same level use the same palette of colors. In particular, if $\mathcal{A}_I$ is on level i, it uses the C colors in the range from $i \cdot C + 1$ to $(i+1)C$.

We determine the input to each $\mathcal{A}_I$ as follows. If I is on level 0, the input to $\mathcal{A}_I$ is simply the entire graph. Otherwise, we decide the input based on the update sequence. For each vertex v, we keep track of its *recent degree*, defined as the number of edges incident to v that were inserted since the last time v was included as input to an instance of the static algorithm. For each interval I, we let v_I be the vertex of highest recent degree at the end of interval I (breaking ties arbitrarily). For the deterministic algorithm, the input to each $\mathcal{A}_I$ is the set $\{v_{I'} | I' \text{ is a subinterval of } I\}$ (where an interval is considered a subinterval of itself).

For the randomized algorithm, in addition to v_I we select another vertex u_I at the end of each interval I. Specifically, we pick uniformly at random an edge insertion (y, z) from the last ℓ updates (if one exists) and then we let u_I be either y or z, chosen at random. Then the input to each $\mathcal{A}_I$ is the set $\{v_{I'} \cup u_{I'} | I' \text{ is a subinterval of } I\}$.

We note that each interval on level $\log_2 n$ contains only 1 subinterval (itself), and generally, each interval on level i contains $n/2^i$ subintervals. Thus, each $\mathcal{A}_I$ on level i takes $O(n/2^i)$ vertices as input.

2.3 Applying the black-box dynamic algorithm

We apply the black-box dynamic algorithm on the graph with the full vertex set but only the edges that count towards the recent degree of both of its endpoints. Specifically, if G denotes the input dynamic graph then the dynamic graph G' that we input to the black-box dynamic algorithm is defined as follows. G' is initially the empty graph on the same vertex set as G and whenever there is an edge update to G, the same edge is updated in G'. Additionally, when a vertex v is included as input to the static algorithm, every edge incident to v is deleted from G'.

To apply the black-box dynamic algorithm, we need to show that G' has bounded maximum degree. To do this, we apply an online 2-player balls and bins game. The game begins with N empty bins. The goal of Player 1 is to maximize the size of the largest bin and the goal of Player 2 is the opposite. At each step, the players each make a move according the following rules.

- Player 1 distributes at most k new balls to its choice of bins.
- Player 2 removes all of the balls from the largest bin (breaking ties arbitrarily).

▶ **Theorem 2.2** ([19]). *In the balls and bins game, every bin always contains $O(k \log N)$ balls.*

A randomized variant of the game will be useful in analyzing our randomized algorithm. In this variant, in addition to emptying the largest bin, Player 2 also chooses a number i from $[k]$ uniformly at random and empties the bin to which Player 1 added its i^{th} ball during its last turn. Player 1 is oblivious to the behavior of Player 2.

▶ **Theorem 2.3** ([20]). *In the randomized variant of the balls and bins game, in a game with N moves every bin always contains $O(k \log\log N + k \log k)$ balls with high probability.*[1]

Recall that ℓ is a parameter introduced in Section 2.1.

▶ **Lemma 2.4.** *In the deterministic algorithm the maximum degree of G' is always $O(\ell \log n)$. In the randomized algorithm the maximum degree of G' is always $O(\ell \log\log n + \ell \log \ell)$.*

Proof. We will argue that in the balls and bins game with $N = n$ and $k = 2\ell$, the number of balls in the largest bin is an upper bound for the maximum degree of G'. Then, applying Theorems 2.2 and 2.3 completes the proof.

We first note that by construction, the degree of each vertex v in G' is at most the recent degree of v so it suffices to bound recent degree. (In particular, the recent degree of v could be larger because it counts edges to vertices that have recently been included as input to the static algorithm.)

The only way for the recent degree of a vertex v to increase is due to the insertion of an edge incident to v. On the other hand, the recent degree of a vertex v decreases when a) an edge incident to v is deleted causing its recent degree to decrement, and b) v is included as input to the static algorithm causing its recent degree to be set to 0.

We consider the special case of the balls and bins game where for each edge insertion (u, v), Player 1 places one ball in the bin corresponding to u and one ball in the bin corresponding to v. Then, when each interval ends (which happens once every ℓ updates), Player 2 moves. Recall that at this point the recent degree of v_I is set to 0 (and in the randomized algorithm, so is that of u_I). It is clear from this description that the balls and bins game parallels all of the increases and some of the decreases in recent degree in the algorithm. From here, it is easy to verify that the number of balls in the largest bin is an upper bound for the maximum recent degree in both the deterministic and randomized settings. We omit the formal proof of this fact due to space constraints. ◀

2.4 Correctness

We will show that our algorithm produces a proper coloring after every update. Recall that the color of each vertex v is defined by the pair of colors (c_1, c_2) where c_1 is the color assigned to v by the black-box static algorithm and c_2 is the color assigned to v by the black-box dynamic algorithm.

Consider an edge (u, v) in the graph at a fixed point in time. We will show that our algorithm assigns different colors to u and v. If (u, v) is included as input to the black-box dynamic algorithm (i.e. if (u, v) is in G'), then its two endpoints are assigned different colors by this algorithm, and are thus assigned different colors by the overall algorithm.

Otherwise, by the definition of the input to the black-box dynamic algorithm, after the edge (u, v) was last inserted at least one of u or v was included as input to the static algorithm. We claim that u and v are assigned different colors by the static algorithm. If u and v were last colored by the same instance $\mathcal{A}_I$ of the static algorithm, then $\mathcal{A}_I$ was executed after the edge (u, v) was inserted (by assumption). Thus, the edge (u, v) was included as input to $\mathcal{A}_I$, causing u and v to be assigned different colors. If u and v were last colored by instances of the static algorithm on different levels, then they are assigned different colors since each level uses a separate palette of colors.

[1] "High probability" means that for all $c > 0$, there is an N such that the probability is at least $1 - N^{-c}$

The only remaining case is that u and v were last included as input to the static algorithm by two different instances of the static algorithm on the same level i. We will show that this is impossible. This case is the crux of the correctness argument and the reason that we define the intervals in precisely the way that we do. It cannot be the case that $i = 0$ since every vertex is recolored at the end of every interval on level 0. Suppose by way of contradiction that u was most recently colored by $\mathcal{A}_I$ (the instance of the static algorithm at the end of interval I) and v was most recently colored by $\mathcal{A}_{I'}$ where interval I comes before interval I' and both are on level i. We will show that between the end of interval I and the end of interval I', u is recolored by an instance of the static algorithm on a level $j < i$ (a contradiction). By the construction of the intervals (see Figure 1), between the ending points of I and I' is the end of an interval I'' on a level $j < i$ that contains interval I as a subinterval. By the definition of the algorithm, every vertex that is included as input to $\mathcal{A}_I$ is also included as input to $\mathcal{A}_{I''}$. Thus, u is recolored on level j before $\mathcal{A}_{I'}$ was executed, a contradiction.

2.5 Analysis

Static algorithm. *Number of colors.* The static algorithm uses C colors per level and there are $O(\log n)$ levels for a total of $O(C \log n)$ colors.
Number of recolorings. In the deterministic algorithm, each interval I has an associated vertex v_I and in the randomized algorithm, each interval has two associated vertices v_I and u_I. Each such vertex is included as input to the static algorithm for all superintervals of I. Since there are $O(\log n)$ levels and each level consists of a set of disjoint intervals, each interval has at most $O(\log n)$ superintervals. Thus, for each interval I, v_I and u_I are included as input to $O(\log n)$ instances of the static algorithm. Every interval ends after a multiple of ℓ updates so the number of recolorings is amortized $O(\frac{\log n}{\ell})$.

Dynamic algorithm. *Number of colors.* Given a dynamic graph of maximum degree Δ, the black-box dynamic algorithm maintains an $O(\Delta)$-coloring. By Lemma 2.4, G' (the graph input to the black-box dynamic algorithm) has maximum degree $O(\ell \log n)$ in the deterministic setting and $O(\ell \log\log n + \ell \log \ell)$ in the randomized setting. The randomized bound is with high probability and in the low probability event that the maximum degree exceeds the bound, we will immediately end all intervals, thereby recoloring the entire graph. Thus, the runtime bound is probabilistic but the bound on the number of colors is not.
Number of recolorings. Using the following simple greedy algorithm we can get constant number of recolorings per update. When an edge is added between two vertices of the same color, simply scan the neighborhood of one of them and recolor it with a non-conflicting color. If the maximum degree of the graph is Δ, this algorithm produces a $\Delta + 1$ coloring.

Combining the static and dynamic algorithms. *Number of colors.* Recall that if a vertex v is assigned color c_1 by the black-box static algorithm and color c_2 by the black-box dynamic algorithm, then our algorithm assigns v the color (c_1, c_2). So the number of colors is the product of the number of colors used in each black-box algorithm, which is $O(C\ell \log^2 n)$ for the deterministic algorithm and $O(C\ell \log n(\log\log n \log \ell))$ for the randomized algorithm.
Number of recolorings. The total number of recolorings is the sum of the number of recolorings performed in each of the black-box algorithms, which is $O(\frac{\log n}{\ell})$. Setting $\ell = \frac{\log n}{\beta}$ completes the proof.

3 Algorithm for low arboricity graphs

In this section we prove Theorem 1.6. The bulk of the argument is to prove Theorem 1.8. We omit the proof of Theorem 1.5.

▶ **Theorem 3.1** (Restatement of Theorem 1.6). *There is a fully dynamic deterministic algorithm for graphs of arboricity α that maintains an $O(\alpha \log^2 n)$-coloring in amortized $\tilde{O}(1)$ time.*

Given a partition of the vertices of a graph into layers $L_1, L_2, \ldots$, for all vertices v let $d_{up}(v)$ (the *up-degree* of v) be the number of neighbors of v in layers equal to or higher than that of v.

▶ **Definition 3.2.** Given a dynamic graph G of arboricity α, a *layered data structure (LDS)* with parameters k and Δ is a partition of the vertices into k layers $L_1, \ldots, L_k$ so that for all vertices v, $d_{up}(v) \le \Delta$.

▶ **Theorem 3.3** (Restatement of Theorem 1.8). *Let A'' be an algorithm for arboricity α graphs that maintains an orientation of the edges with out-degree at most D that performs amortized $F(n)$ flips per update. Then there is an algorithm to maintain an LDS for a fully dynamic graph of arboricity α with $k = O(\log n)$ and $\Delta = O(D + \alpha \log n)$ in amortized deterministic time $O(F(n))$.*

3.1 Proof overview

The idea of the algorithm is essentially to move vertices to new layers when the required properties of the data structure are violated. Roughly, when there is a vertex v with $d_{up}(v) \ge \Delta$ we move v to a higher layer so that $d_{up}(v)$ decreases to $O(\alpha)$. Additionally, to control the number of layers, whenever a vertex v has up-degree less than $d = O(\alpha)$ and v can be moved to a lower layer while maintaining up-degree less than d, we move v to a lower layer. The fact that d and Δ differ by a logarithmic factor ensures that vertices don't move between layers too often which is essential for bounding the runtime.

To help with the runtime analysis, we maintain *two* dynamic orientations of the edges: one is defined by the algorithm A'' and the other is maintained by our algorithm and ensures that all edges with endpoints in different layers are oriented toward the higher layer. We compare the number of edge flips in the orientation defined by our algorithm to the number of edge flips in the orientation algorithm A'' using a potential function: $\phi(i) =$ the number of oppositely oriented edges in the two algorithms. This potential function is also used in [14].

The main idea of the analysis is to observe how ϕ changes in response to vertices moving between levels. We claim that when we move a vertex to a higher level, ϕ decreases substantially. Our algorithm is defined so that we only move a vertex to a higher layer if its up-degree decreases substantially as a result. Because our algorithm orients edges from lower to higher layers, when we move a vertex v to a higher layer many edges incident to v are flipped towards v. Then because A'' maintains an orientation of low out-degree, many of these edges flipped towards v end up oriented in the same direction in the two orientations. Thus, ϕ decreases substantially as a result of v moving to a higher layer. On the other hand, when a vertex moves to a lower layer, ϕ might increase. The idea of the argument is to use the substantial decreases in ϕ that result from moving vertices to higher layers to pay for the increases in ϕ that result from moving vertices to lower layers.

3.2 Invariants

In this section we introduce four invariants that together imply that $d_{up}(v) \leq \Delta$ and $k = O(\log n)$.

We maintain two dynamic orientations of the edges in the graph, one defined by our algorithm and the other defined by the algorithm A''. Unless otherwise stated, when we refer to an orientation, we mean the orientation defined by our algorithm.

For ease of notation, let $d = 4\alpha$ and let $d' = \Delta/2$.
We define the following for each vertex v:

- $L(v)$ is the layer containing v.
- $L_{max}(v)$ is the lowest layer for which if v were in this layer, $d_{up}(v)$ would be at most d.
- $d^+(v)$ is the out-degree of v.
- $d_L^-(v)$ is the in-degree of v from neighbors in $L(v)$.

Invariant 1 defines how edges are oriented between layers and is useful for analyzing the update time of the algorithm, as outlined in Section 3.1.

▶ **Invariant 1.** *All edges with endpoints in different layers are oriented towards the vertex in the higher layer.*

The next two invariants bound $d^+(v)$ and $d_L^-(v)$, which helps to bound $d_{up}(v)$.

▶ **Invariant 2.** *For all vertices v, $d^+(v) \leq d'$.*

▶ **Invariant 3.** *For all vertices v, $d_L^-(v) \leq d'$.*

▶ **Claim 3.4.** *Invariants 1-3 together imply that $d_{up}(v) \leq 2d' = \Delta$.*

Proof. By Invariant 1, for all vertices v, every neighbor of v in a layer equal to or higher than $L(v)$ is either an out-neighbor of v or an in-neighbor of v in $L(v)$, so $d_{up}(v) = d^+(v) + d_L^-(v)$. Then by Invariants 2 and 3, $d^+(v) + d_L^-(v) \leq 2d'$. ◀

Invariant 4 serves to bound the number of layers k.

▶ **Invariant 4.** *For all vertices v, $L(v) \leq L_{max}(v)$.*

▶ **Claim 3.5.** *Invariant 4 implies that $k = O(\log n)$.*

Proof. First we observe that under Invariant 4, all vertices of degree at most d are in L_1. Now, consider removing all vertices in L_1 from the graph. In the remaining graph, all vertices of degree at most d are in layer L_2. More generally, after removing all vertices in layers 1 through i for any i, all vertices of degree at most d must be in layer L_{i+1}.

The total number of edges in a graph of arboricity alpha is less than αn. So at least a $(1 - 2\alpha/d)$ fraction of the vertices have degree at most d. Any subgraph of an arboricity α graph also has arboricity α so after the vertices in any given layer are removed, the graph still has arboricity α. Thus, after removing the vertices in layers 1 through i for any i, at least a a $(1 - 2\alpha/d)$ fraction of the remaining vertices are in L_{i+1}. Therefore, the number k of layers total is at most $\log_{\frac{d}{2\alpha}} n = O(\log n)$. ◀

3.3 Algorithm

The idea of the algorithm is essentially to move vertices to new layers when the required properties of the data structure are violated. We define two recursive procedures Rise and Drop which move vertices to higher and lower layers respectively. In particular, when a vertex v violates Invariant 2 or 3 (i.e. either $d^+(v) > d'$ or $d^-_L(v) > d'$), we call the procedure Rise(v) which moves v up to the layer $L_{max}(v)$. The movement of v to a new higher layer may increase the up-degree of some neighbors u of v causing u to violate Invariant 2 or 3, in which case we recursively call Rise(u). On the other hand, when a vertex v violates Invariant 4 (i.e. $L_{max}(v) < L(v)$), we call the procedure Drop(v) which moves v down to the layer $L_{max}(v)$. The movement of v to a new lower layer may decrease $L_{max}(u)$ for some neighbors u of v causing u to violate Invariant 4, in which case we recursively call Drop(u).

We defer the pseudocode of the algorithm, the analysis of the algorithm, and the proof of Theorem 1.6 to the full version.

References

1 Ittai Abraham, Shiri Chechik, and Sebastian Krinninger. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, January 16-19, 2017*, pages 440–452, 2017.

2 A. Andersson and Mikkel Thorup. Dynamic ordered sets with exponential search trees. *J. ACM*, 54(3):13, 2007.

3 Srinivasa R Arikati, Anil Maheshwari, and Christos D Zaroliagis. Efficient computation of implicit representations of sparse graphs. *Discrete Applied Mathematics*, 78(1-3):1–16, 1997.

4 Baruch Awerbuch, Michael Luby, Andrew V Goldberg, and Serge A Plotkin. Network decomposition and locality in distributed computation. In *FOCS, 1989., 30th Annual Symposium on*, pages 364–369. IEEE, 1989.

5 Luis Barba, Jean Cardinal, Matias Korman, Stefan Langerman, André van Renssen, Marcel Roeloffzen, and Sander Verdonschot. Dynamic graph coloring. In *Proceedings of the 15th International Symposium on Algorithms and Data Structures, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017*, pages 97–108, 2017.

6 Leonid Barenboim and Michael Elkin. Sublogarithmic distributed mis algorithm for sparse graphs using nash-williams decomposition. *Distributed Computing*, 22(5-6):363–379, 2010.

7 Leonid Barenboim and Michael Elkin. Distributed graph coloring: Fundamentals and recent developments. *Synthesis Lectures on Distributed Computing Theory*, 4(1):1–171, 2013.

8 Leonid Barenboim and Tzalik Maimon. Fully-dynamic graph algorithms with sublinear time inspired by distributed computing. In *Proceedings of the International Conference on Computational Science, ICCS 2017, Zurich, Switzerland, June 12-14, 2017*, pages 89–98, 2017.

9 Edvin Berglin and Gerth Stølting Brodal. A simple greedy algorithm for dynamic graph orientation. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 92. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

10 Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Part I*, pages 167–179, 2015.

11 Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 692–711, 2016.

12 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1–20, 2018.

13 Greg Bodwin and Sebastian Krinninger. Fully dynamic spanners with worst-case update time. *arXiv preprint arXiv:1606.07864*, 2016.

14 G. S. Brodal and R. Fagerberg. Dynamic representation of sparse graphs. In *Proc. of 6th WADS*, pages 342–351, 1999.

15 Moses Charikar and Shay Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial barrier for worst-case time bounds. *arXiv preprint arXiv:1711.06883*, 2017.

16 Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.

17 Artur Czumaj and Christian Sohler. Testing hypergraph coloring. In *International Colloquium on Automata, Languages, and Programming*, pages 493–505. Springer, 2001.

18 Maximilien Danisch, Oana Balalau, and Mauro Sozio. Listing k-cliques in sparse real-world graphs. *communities*, 28:43, 2018.

19 Paul Dietz and Daniel Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 365–372. ACM, 1987.

20 Paul F Dietz and Rajeev Raman. Persistence, amortization and randomization. In *Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*, pages 78–88. Society for Industrial and Applied Mathematics, 1991.

21 Paul F Dietz and Rajeev Raman. A constant update time finger search tree. *Information Processing Letters*, 52(3):147–154, 1994.

22 Zdeněk Dvořák and Vojtěch Tuma. A dynamic data structure for counting subgraphs in sparse graphs. In *Workshop on Algorithms and Data Structures*, pages 304–315. Springer, 2013.

23 Talya Eden, Reut Levi, and Dana Ron. Testing bounded arboricity. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2081–2092. SIAM, 2018.

24 Guy Even, Moti Medina, and Dana Ron. Deterministic stateless centralized local algorithms for bounded degree graphs. In *European Symposium on Algorithms*, pages 394–405. Springer, 2014.

25 Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni. Fully dynamic shortest paths in digraphs with arbitrary arc weights. *Journal of Algorithms*, 49(1):86–113, 2003.

26 Andrew V Goldberg, Serge A Plotkin, and Gregory E Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4):434–446, 1988.

27 Oded Goldreich, Shari Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998.

28 Bradley Hardy, Rhyd Lewis, and Jonathan Thompson. Modifying colourings between time-steps to tackle changes in dynamic random graphs. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 186–201. Springer, 2016.

29 Bradley Hardy, Rhyd Lewis, and Jonathan Thompson. Tackling the edge dynamic graph colouring problem with and without future adjacency information. *Journal of Heuristics*, pages 1–23, 2017.

30 M. He, G. Tang, and N. Zeh. Orienting dynamic graphs, with applications to maximal matchings and adjacency queries. In *Proc. 25th ISAAC*, pages 128–140, 2014.

31 Shweta Jain and C Seshadhri. A fast and provable method for estimating clique counts using turán's theorem. In *Proceedings of the 26th International Conference on World Wide Web*, pages 441–449. International World Wide Web Conferences Steering Committee, 2017.

32 Alexis Kaporis, Christos Makris, George Mavritsakis, Spyros Sioutas, Athanasios Tsakalidis, Kostas Tsichlas, and Christos Zaroliagis. Isb-tree: a new indexing scheme with efficient expected behaviour. In *International Symposium on Algorithms and Computation*, pages 318–327. Springer, 2005.

33 Subhash Khot and Ashok Kumar Ponnuswami. Better inapproximability results for maxclique, chromatic number and min-3lin-deletion. In *Proc. 33rd ICALP*, pages 226–237, 2006.

34 T. Kopelowitz, R. Krauthgamer, E. Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *Proc. 41st ICALP*, pages 532–543, 2014.

35 Łukasz Kowalik. Fast 3-coloring triangle-free planar graphs. In *European Symposium on Algorithms*, pages 436–447. Springer, 2004.

36 Łukasz Kowalik. Adjacency queries in dynamic sparse graphs. *Information Processing Letters*, 102(5):191–195, 2007.

37 Lukasz Kowalik and Maciej Kurowski. Oracles for bounded-length shortest paths in planar graphs. *ACM Transactions on Algorithms (TALG)*, 2(3):335–363, 2006.

38 Christos Levcopoulos and Mark H. Overmars. A balanced search tree with O (1) worst-case update time. *Acta Inf.*, 26(3):269–277, 1988.

39 Min Chih Lin, Francisco J Soulignac, and Jayme L Szwarcfiter. Arboricity, h-index, and dynamic algorithms. *Theoretical Computer Science*, 426:75–90, 2012.

40 Nathan Linial. Distributive graph algorithms-global solutions from local data. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 331–335, 1987.

41 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

42 Michael Luby. Removing randomness in parallel computation without a processor penalty. *J. Comput. Syst. Sci.*, 47(2):250–286, 1993.

43 Cara Monical and Forrest Stonedahl. Static vs. dynamic populations in genetic algorithms for coloring a dynamic graph. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 469–476. ACM, 2014.

44 Christian Worm Mortensen. Fully dynamic orthogonal range reporting on ram. *SIAM Journal on Computing*, 35(6):1494–1525, 2006.

45 C St JA Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, 1(1):12–12, 1964.

46 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Proceedings of the 45th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2013, Palo Alto, CA, USA, June 1-4, 2013*, pages 745–754, 2013.

47 K. Onak, B. Schieber, S. Solomon, and N. Wein. Fully dynamic mis in uniformly sparse graphs. In *Proc. 45th ICALP*, 2018.

48 Linda Ouerfelli and Hend Bouziri. Greedy algorithms for dynamic graph coloring. In *Communications, Computing and Control Applications (CCCA), 2011 International Conference on*, pages 1–5. IEEE, 2011.

49 Merav Parter, David Peleg, and Shay Solomon. Local-on-average distributed tasks. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 220–239, 2016.

50 David Peleg and Shay Solomon. Dynamic $(1 + \epsilon)$-approximate matchings: A density-sensitive approach. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, 2016.

51 Davy Preuveneers and Yolande Berbers. Acodygra: an agent algorithm for coloring dynamic graphs. *Symbolic and Numeric Algorithms for Scientific Computing (September 2004)*, 6:381–390, 2004.

52 Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *Proc. of 1st ITCS*, pages 223–238, 2011.

53 Scott Sallinen, Keita Iwabuchi, Suraj Poudel, Maya Gokhale, Matei Ripeanu, and Roger Pearce. Graph colouring as a challenge problem for dynamic graph processing on distributed systems. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*, pages 347–358. IEEE, 2016.

54 Stephen B Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

55 Shay Solomon. Fully dynamic maximal matching in constant update time. In *Proceedings of the 57th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2016, New Brunswick, NJ, USA, October 9-11, 2016*, pages 325–334, 2016.

56 Shay Solomon. Local algorithms for bounded degree sparsifiers in sparse graphs. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 94. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

57 Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proceedings of the 37th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2005, Baltimore, MD, USA, May 21-24, 2005*, pages 112–119, 2005.

58 Christian Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1130–1143, 2017.

59 Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. Effective and efficient dynamic graph coloring. *Proceedings of the VLDB Endowment*, 11(3):338–351, 2017.

60 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007.

Soft Subdivision Motion Planning for Complex Planar Robots

Bo Zhou
Department of Computer Science and Engineering, New York University, Brooklyn, NY, USA
bz387@nyu.edu

Yi-Jen Chiang
Department of Computer Science and Engineering, New York University, Brooklyn, NY, USA
chiang@nyu.edu

Chee Yap
Department of Computer Science, New York University, New York, NY, USA
yap@cs.nyu.edu

Abstract

The design and implementation of theoretically-sound robot motion planning algorithms is challenging. Within the framework of *resolution-exact algorithms*, it is possible to exploit *soft predicates* for collision detection. The design of soft predicates is a balancing act between easily implementable predicates and their accuracy/effectivity.

In this paper, we focus on the class of planar polygonal rigid robots with arbitrarily complex geometry. We exploit the remarkable *decomposability* property of soft collision-detection predicates of such robots. We introduce a general technique to produce such a decomposition. If the robot is an m-gon, the complexity of this approach scales linearly in m. This contrasts with the $O(m^3)$ complexity known for exact planners. It follows that we can now routinely produce soft predicates for any rigid polygonal robot. This results in resolution-exact planners for such robots within the general *Soft Subdivision Search* (SSS) framework. This is a significant advancement in the theory of sound and complete planners for planar robots.

We implemented such decomposed predicates in our open-source `Core Library`. The experiments show that our algorithms are effective, perform in real time on non-trivial environments, and can outperform many sampling-based methods.

2012 ACM Subject Classification Theory of computation → Computational geometry, Computing methodologies → Robotic planning

Keywords and phrases Computational Geometry, Algorithmic Motion Planning, Resolution-Exact Algorithms, Soft Predicates, Planar Robots with Complex Geometry

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.73

Related Version A full version of the paper is available at [20], `http://cse.poly.edu/chiang/esa18-full.pdf`.

1 Introduction

Motion planning is widely studied in robotics [9, 10, 5]. Many planners are heuristics, i.e., without a priori guarantees of its performance. In this paper, we are interested in non-heuristic algorithms for the **basic planning problem**: this basic problem involves only kinematics and the existence of paths. The robot R_0 is fixed, and the input is a triple (α, β, Ω) where α, β are the start and goal configurations of R_0, and $\Omega \subseteq \mathbb{R}^d$ is a polyhedral

© Bo Zhou, Yi-Jen Chiang, and Chee Yap;
licensed under Creative Commons License CC-BY

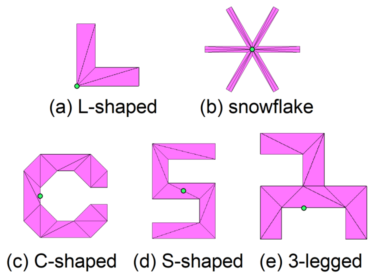
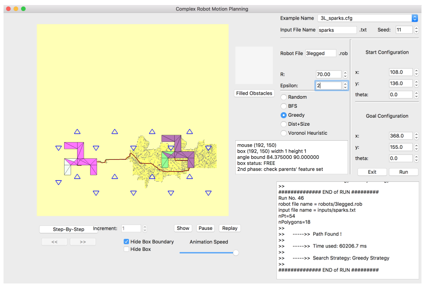


Figure 1 Some rigid planar robots ((a)-(b): star-shaped; (c)-(e): general shaped).



Example Name: 3L_sparks.cfg
Input File Name: sparks .txt
Seed: 11
Robot File: 3legged .rob
R: 70.00
Epsilon: 2
Random
BFS
[✓ Greedy]
Dist+Size
Voronoi Heuristic
Start Configuration x: 108.0 y: 136.0 theta: 0.0
Goal Configuration x: 368.0 y: 155.0 theta: 0.0
Increment: 1
[✓ Hide Box Boundary]
Hide Box

```
mouse (192, 150)
box (192, 150) width 1 height 1
angle bound 84.375000 90.000000
box status: FREE
2nd phase: check parents' feature set
```

```
>>
############## END of RUN #########
Run No. 46
robot file name = robots/3legged.rob
input file name = inputs/sparks.txt
nPt=54
nPolygons=18
>>
>>     ----->>  Path Found !
>>
>>     ----->>  Time used: 60206.7 ms
>>
>>     ----->>  Search Strategy: Greedy Strategy
>>
############## END of RUN #########
```

Figure 2 GUI interface for planner for a 3-legged robot.

environment in $d = 2$ or 3. The algorithm outputs an Ω-avoiding path from α to β if one exists, and NO-PATH otherwise. See Figure 1 for some rigid robots, and also Figure 2 for our GUI interface for path planning.

The basic planning problem ignores issues such as the optimality of paths, robot dynamics, planning in the time dimension, non-holonomic constraints, and other considerations of a real scenario. Despite such an idealization, the solution to this basic planning problem is often useful as the basis for finding solutions that do take into account the omitted considerations. E.g., given a kinematic path, we can plan a smooth trajectory with a homotopic trace.

The algorithms for this basic problem are called "planners". In theory, it is possible to design *exact* planners because the basic path planning is a semi-algebraic (non-transcendental) problem. Even when such algorithms are available, exact planners have relatively high complexity and are non-adaptive, even in the plane (see [12]). So we tend to see inexact implementations of exact algorithms, with unclear guarantees. When fully explicit algorithms are known, exact implementation of exact planners is possible using suitable software tools such as the CGAL library [7].

In current robotics [10, 5], those algorithms that are considered practical and have some guarantees may be classified as either resolution-based or sampling-based. The guarantees for the former is the notion of **resolution completeness** and for the latter, **sampling completeness**. Roughly speaking, *if there exists a path* then:

- resolution completeness says that a path will be found if the resolution is fine enough;
- sampling completeness says that a path will be found with high probability if "enough" random samples are taken.

But notice that if there is no path, these criteria are silent; indeed, such algorithms would not halt except by artificial cut-offs. Thus a major effort in the last 20 years of sampling research has been devoted to the so-called "Narrow Passage" problem. It is possible to view this problem as a manifestation of the **Halting Problem** for the sampling approaches: how can the algorithm halt when there is no path? (A possible approach to address this problem might be to combine sampling with exact computation, as in [13].)

Motivated by such issues, as well as trying to avoid the need for exact computation, we in [15, 17] introduced the following replacement for resolution complete planners: a **resolution-exact planner** takes an extra input parameter $\epsilon > 0$ in addition to (α, β, Ω), and it always halts and outputs either an Ω-avoiding path from α to β or NO-PATH. The output satisfies this condition: there is a constant $K > 1$ depending on the planner, but independent of the inputs, such that:

- if there is a path of clearance $K\epsilon$, it must output a path;
- if there is no path of clearance ϵ/K, it must output NO-PATH.

Notice that if the optimal clearance lies between $K\epsilon$ and ϵ/K, then the algorithm may output either a path or `NO-PATH`. So there is output indeterminacy. Note that the traditional way of using ϵ is to fix $K = 1$, killing off indeterminacy. Unfortunately, this also leads us right back to exact computation which we had wanted to avoid. We believe that indeterminacy is a small price to pay in exchange for avoiding exact computation [15]. The practical efficiency of resolution-exact algorithms is demonstrated by implementations of planar robots with 2, 3 and 4 degrees of freedom (DOF) [15, 11, 16], and also 5-DOF spatial robots [8]. All these robots perform in real-time in non-trivial environments. In view of the much stronger guarantees of performance, resolution-exact algorithms might reasonably be expected to have a lower efficiency compared to sampling algorithms. Surprisingly, no such trade-offs were observed: resolution-exact algorithms consistently outperform sampling algorithms. Our 2-link robot [11, 16] was further generalized to have thickness (a feat that exact methods cannot easily duplicate), and can satisfy a non-self-crossing constraint, all without any appreciable slowdown. Finally, these planners are more general than the basic problem: they all work for parametrized families $R_0(t_1, t_2 \ldots)$ of robots, where t_i's are robot parameters. All these suggest the great promise of our approach.

What is new in this paper. In theoretical path planning, the algorithms often focused on simple robots like discs or line segments. In this paper, we address the issue of "complex robots" where the complexity comes from the geometry of the robots rather than from the degrees of freedom. Complex robots provide more realistic models for real-world robots. We focus on planar robots that are rigid and connected. Such a robot may be represented by a compact connected polygonal set $R_0 \subseteq \mathbb{R}^2$ whose boundary is an m-sided polygon, i.e., an m-gon. Informally, we call R_0 a "complex robot" if it is a non-convex m-gon for "moderately large" values of m, say $m \geq 5$. By this criterion, all the robots in Figure 1 are "complex". According to [19], no exact algorithms for $m > 3$ have been implemented; in this paper, we have robots with $m = 18$. To see why complex robots may be challenging, recall that the free space of such robots may have complexity $O((mn)^3 \log(mn))$ (see [1]) when the robot and environment have complexity m and n, respectively. Even with m fixed, this can render the algorithm impractical. For instance, if $m = 10$, the algorithm may slow down by 3 orders of magnitude. But our subdivision approach does not have to compute the entire free space before planning a path; hence the worst-case cubic complexity of the free space is not necessarily an issue.

More importantly, we show that the complexity of our new method grows only linearly with m. To achieve this, we exploit a remarkable property of soft predicates called "decomposability". We show how an arbitrary complex robot can be decomposed (via triangulation that may introduce new vertices) into an ensemble of "nice triangles" for which soft predicates are easy to implement. As we see below, there is a significant difference between a single triangle and an ensemble of triangles. *In consequence of our new techniques, we can now routinely construct resolution-exact planners for any reasonably complex robot provided by a user.* This could lead to a flowering of experimentation algorithmics in this subfield.

Technically, it is important to note that the previous soft predicate construction for a triangle robot in [15, 18] requires that the rotation center, i.e., the origin of the (rotational) coordinate system, be chosen to be the circumcenter of the triangle. But for our new soft predicates the triangles in the triangulation of the complex robot cannot be treated in the same way. This is because all the triangles of the triangulation must share a **common origin**, to serve as the rotation center of the robot. To ensure easy-to-compute predicates, we introduce the notion of a "nice triangulation" relative to a chosen origin: all triangles must be "nice" relative to this origin. These ideas apply for arbitrary complex robots, but we also exploit the special case of star-shaped robots to achieve stronger results.

Figure 2 shows our experimental setup for complex robots. A demo showing the real-time performance of our algorithms is found in the video clip available through this web link: `https://cs.nyu.edu/exact/gallery/complex/complex-robot-demo.mp4`. All proofs are deferred to the full version of this paper [20].

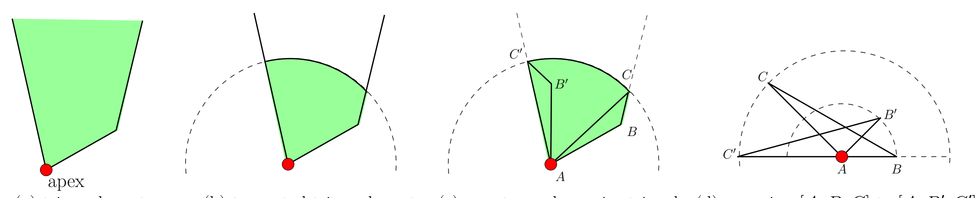
Remark. Although it is not our immediate concern to address noisy environments and uncertainties, it is clear that our work can be leveraged to address these issues. E.g., users can choose $\epsilon > 0$ to be correlated with the uncertainty in the environment and the precision of the robot sensors. By using weighted Voronoi diagrams [4], we can achieve practical planners that have obstacle-dependent clearances (larger clearance for "dangerous" obstacles).

Previous related work. An early work is Zhu-Latombe [21] who also classify boxes into FREE or MIXED or STUCK (using our terminology below). They introduced the concept of **M-channels** (comprised of FREE or MIXED leaf boxes), as a heuristic basis to find an F-channel comprising only of FREE boxes. Subsequent researchers (Barbehenn-Hutchinson [2] and Zhang-Manocha-Kim [19]) continued this approach. Researchers in resolution-based approaches were interested in detecting the non-existence of paths, but their solutions remain partial because they do not guarantee to always detect non-existence of paths (of sufficient clearances) [3, 19]. The challenge of complex robots was taken up by Manocha's group who implemented a series of such examples [19]: a "five-gear" robot, a "2-D puzzle" robot a certain "star" robot with 4 DOFs, and a "serial link" robot with 4 DOFs. Except for the "star", the rest are planar robots.

2 Review: Fundamentals of Soft Subdivision Approach

Our soft subdivision approach includes the following three fundamental concepts (see [15] and the Appendix of [11] for the details):

- Resolution-exactness. This is an alternative replacement for the standard concept of "resolution completeness" in the subdivision literature. Briefly, a planner is **resolution-exact** if there is a constant $K > 1$ such that if there is a path of clearance $K\epsilon$, it will return a path, and if there is no path of clearance ϵ/K, it will return NO-PATH. Here, $\epsilon > 0$ is an additional input to the planner, in addition to the normal parameters.
- Soft Predicates. Let $\square\mathbb{R}^d$ be the set of closed axes-aligned boxes in $\mathbb{R}^d$. We are interested in predicates that classify boxes. Let $C : \mathbb{R}^d \to \{+1, 0, -1\}$ be an (exact) predicate where $+1, -1$ are called definite values, and 0 the indefinite value. For motion planning, we may also identify $+1/-1/0$ with FREE/STUCK/MIXED, respectively. In our application, if p is a free configuration, then $C(p) =$ FREE; if p is on the boundary of the free space, $C(p) =$ MIXED; otherwise $C(p) =$ STUCK. We extend C to boxes $B \in \square\mathbb{R}^d$ as follows: for a definite value $v \in \{+1, -1\}$, $C(B) := v$ if $C(x) = v$ for every $x \in B$. Otherwise, $C(B) := 0$. Call $\widetilde{C} : \square\mathbb{R}^d \to \{+1, 0, -1\}$ a "soft version" of C if whenever $\widetilde{C}(B)$ is a definite value, $\widetilde{C}(B) = C(B)$, and moreover, if for any sequence of boxes B_i $(i \geq 1)$ that converges monotonically to a point p, $\widetilde{C}(B_i) = C(p)$ for i large enough.
- Soft Subdivision Search (SSS) Framework. This is a general framework for a broad class of motion planning algorithms. One must supply a small number of subroutines with fairly general properties in order to derive a specific algorithm. For SSS, we need a predicate to classify boxes in the configuration space as FREE/STUCK/MIXED, a method to split boxes, a method to test if two FREE boxes are connected by a path of FREE boxes, and a method to pick MIXED boxes for splitting. The power of such frameworks is that we can explore a great variety of techniques and strategies. Indeed we introduced the SSS framework to emulate such properties found in the sampling framework.



(a) triangular set (unbounded case) (b) truncated triangular set (c) swept area by a nice triangle (d) sweeping $[A, B, C]$ to $[A, B', C']$

Figure 3 Truncated triangular set and swept areas.

Feature-Based Approach. Following our previous work [15, 11], our computation and predicates are "feature based" whereby the evaluations of box primitives are based on a set $\widetilde{\phi}(B)$ of features associated with the box B. Given a polygonal set $\Omega \subseteq \mathbb{R}^2$ of obstacles, the boundary $\partial\Omega$ may be subdivided into a unique set of **corners** (points) and **edges** (open line segments), called the **features** of Ω. Let $\Phi(\Omega)$ denote this feature set. Our representation of $f \in \Phi(\Omega)$ ensures this **local property of** f: *for any point q, if f is the closest feature to q, then we can decide if q is inside Ω or not.* To see this, first note that if f is a corner, then q is outside Ω iff f is a convex corner of Ω. But if f is an edge, our representation assigns an orientation to f such that q is inside Ω iff q lies to the left of the oriented line through f.

3 Star-Shaped Robots

We first consider star-shaped robots. A star-shaped region R is one for which there exists a point $A \in R$ such that any line through A intersects R in a single line segment. We call A a **center** of R. Note that A is not unique. When a robot R_0 is a star-shaped polygon, we decompose R_0 into a set of triangles that share a **common vertex** at a center A. The rotations of the robot R_0 about the point A can then be reduced to the rotations of "nice" triangles about A. The soft predicates of nice triangles will be easy to implement because their footprints have special representations.

3.1 Nice Shapes for Rotation

From now on, by a **triangular set** we mean a subset $T \subseteq \mathbb{R}^2$ which is written as the non-redundant intersection of three closed half-spaces: $T = H_1 \cap H_2 \cap H_3$. Non-redundant means that we cannot express T as the intersection of only two half-spaces. Note that if T is bounded, this is our familiar notion of a triangle with 3 vertices. But T might be unbounded and have only 2 vertices as in Figure 3(a). If T is a triangular set, we may arbitrarily call one of its vertices the **apex** and call the resulting T a **pointed triangular set**. By a **truncated triangular set (TTS)**, we mean the intersection of a pointed triangular set T with any disc centered at its apex A, as shown in Figure 3(b).

Notation for Angular Range: It is usual to identify S^1 (unit circle) with the interval $[0, 2\pi]$ where 0 and 2π are identified. Let $\alpha \neq \beta \in S^1$. Then $[\alpha, \beta]$ denote the range of angles from α counter-clockwise to β. Thus $[\alpha, \beta]$ and $[\beta, \alpha]$ are complementary ranges in S^1. If $\Theta = [\alpha, \beta]$, then its **width**, $|\Theta|$ is defined as $\beta - \alpha$ if $\beta > \alpha$, and $2\pi + \beta - \alpha$ otherwise. Moreover, we will write "$\alpha < \theta < \beta$" to mean that $\theta \in [\alpha, \beta]$.

Fix an arbitrary bounded triangular set T_0, represented by its three vertices A, B, C where A is the apex. For $\theta \in S^1$, let $T_0[\theta]$ denote the footprint of T_0 after rotating T_0 counter-clockwise (CCW) by θ about the apex. If $\Theta \subseteq S^1$, we write $T_0[\Theta] = \bigcup\{T_0[\theta] : \theta \in \Theta\}$. The sets $T_0[\theta]$ and $T_0[\Theta]$ are called **footprints** of T_0 at θ and Θ, respectively. If $\Theta = [\alpha, \beta]$, write $T_0[\alpha, \beta]$ for $T_0[\Theta]$, and call $T_0[\alpha, \beta]$ the **swept area** as T_0 rotates from α to β.

One of our concerns is to ensure that the swept area $T_0[\Theta]$ is "nice". Consider an example where $[A, B, C]$ is a triangular set with apex A (see Figure 3(c)). Consider the area swept by rotating $[A, B, C]$ in a CCW direction about its apex to position $[A, B', C']$. This sweeps out the truncated triangular set shown in Figure 3(b). This truncated triangular set (TTS) is desirable since it can be easily specified by the intersection of three half-spaces and a disc. On the other hand, if $[A, B, C]$ is the triangular set in Figure 3(d), then no rotation of $[A, B, C]$ would sweep out a truncated triangular set. So the triangular set in Figure 3(d) is "not nice", unlike the triangular set in Figure 3(c).

In general, let $T = [A, B, C]$ be a bounded triangular set. Let a, b, c denote the corresponding angles at A, B, C. We say T is **nice** if either b or c is at least $\pi/2$ ($= 90°$). We call the corresponding vertex (B or C) a **nice vertex**. Assuming T is non-degenerate and nice, there is a unique nice vertex. In the following, we assume (w.l.o.g.) that B is the nice vertex. The reason for defining niceness is the following.

▶ **Lemma 1.** *Let T be a pointed triangular set. Then T is nice iff for all $\alpha \in S^1$ ($0 < \alpha < \pi - a$), the footprints $T[0, \alpha]$ and $T[-\alpha, 0]$ are truncated triangular sets (TTS).*

▶ **Lemma 2.** *Let R_0 be a star-shaped polygonal region with A as center. If the boundary of R_0 is an n-gon, then we can decompose R_0 into an essentially disjoint[1] union of at most $2n$ bounded triangular sets (i.e., at most $2n$ triangles) that are **nice** and have A as the apex.*

3.2 Complex Predicates and T/R Subdivision Scheme

For complex robots in general (not necessarily star-shaped), we can exploit the remarkable **decomposability** property of soft predicates. More specifically, suppose $R_0 = \cup_{j=1}^{m} T_j$ where each T_j is a triangle or other shapes and not necessarily pairwise disjoint. If we have soft predicates $\widetilde{C}_j(B)$ for each T_j (where B is a box), then we immediately obtain a soft predicate for R_0 defined as follows:

$$\widetilde{C}(B) = \begin{cases} \texttt{FREE} & \text{if each } \widetilde{C}_j(B) \text{ is } \texttt{FREE} \\ \texttt{STUCK} & \text{if some } \widetilde{C}_j(B) \text{ is } \texttt{STUCK} \\ \texttt{MIXED} & \text{otherwise.} \end{cases} \tag{1}$$

Let $\sigma > 1$ and $\widetilde{C}$ be the soft version of an exact predicate C. Recall [15, 18] that $\widetilde{C}$ is **σ-effective** if for all boxes B, if $C(B) = \texttt{FREE}$ then $\widetilde{C}(B/\sigma) = \texttt{FREE}$.

Proposition A.
(1) *$\widetilde{C}$ is a soft version of the exact classification predicate for R_0.*
(2) *Moreover, if each $\widetilde{C}_j$ is σ-effective, then $\widetilde{C}$ is σ-effective.*

We need σ-effectivity in soft predicates in order to ensure resolution-exactness; see [15, 18] where this proposition was proved. There are two important remarks. First, this proposition is **false** if the $\widetilde{C}_j$ and $\widetilde{C}$ were exact predicates. More precisely, suppose C is the exact predicate for R_0 and C_j is the exact predicate for each T_j. It is true that if $C(B) = \texttt{FREE}$ then $C_j(B) = \texttt{FREE}$ for all j. But if $C(B) = \texttt{STUCK}$, it does not follow that $C_j(B) = \texttt{STUCK}$ for some j. Second, the predicates $\widetilde{C}_j(B)$ for all the T_j's must be based on a **common**

[1] A set $\{A_1, \ldots, A_k\}$ where each $A_i \subseteq \mathbb{R}^2$ is said to be **essentially disjoint** if the interiors of the A_i's are pairwise disjoint.

coordinate system. As mentioned in Sec. 1, the soft predicate construction for a triangle robot in [15] does not work here. A technical contribution of this paper is the design of soft predicates $\widetilde{C}_j(B)$ for all the T_j's that are based on a common coordinate system. In the case of star-shaped robots, we apply Lemma 2 and use the apex A as the origin of this common coordinate system. Let r_j be the length of the longer edge out of A in T_j. We define r_0 as $r_0 = \max_j r_j$ (i.e., r_0 is the radius of the circumcircle of R_0 centered at A).

T/R Splitting. The simplest splitting strategy is to split a box $B \subseteq \mathbb{R}^d$ into 2^d congruent subboxes. In the worst case, to reduce all boxes to size $< \epsilon$ requires time $\Omega(\log(1/\epsilon)^d)$; this complexity would not be practical for $d > 3$. In [11, 16] we introduced an effective solution called *T/R splitting* which can be adapted to configuration space[2] $SE(2)$ in the current paper. Write a box $B \subseteq SE(2)$ as a pair (B^t, B^r) where $B^t \subseteq \mathbb{R}^2$ is the translational box and $B^r \subseteq S^1$ an angular range Θ. We say box $B = (B^t, B^r)$ is **ε-small** if B^t and B^r are both ε-small; the former means the width of B^t is $\leq \varepsilon$; the latter means the angle (in radians) satisfies $|B^r| \leq \varepsilon/r_0$. Our splitting strategy is to only split B^t (leaving $B^r = S^1$) as long as B^t is not ε-small. This is called a **T-split**, and produces 4 children. Once B^t is ε-small, we do binary splits of B^r (called **R-split**) until B^r is ε-small. We discard B when it is ε-small. The following lemma (and proof) in [15] can be carried over here:

▶ **Lemma 3.** *([15]) Assume $0 < \varepsilon \leq \pi/2$. If $B = (B^t, B^r)$ is ε-small and B^t is a square, then the Hausdorff distance between the footprints of R_0 at any two configurations in B is at most $(1 + \sqrt{2})\varepsilon$.*

Soft Predicates. Suppose we want to compute a soft predicate $\widetilde{C}(B)$ to classify boxes B. Following the previous work [15, 11], we reduce this to computing a feature set $\widetilde{\phi}(B) \subseteq \Phi(\Omega)$. The **feature set** $\widetilde{\phi}(B)$ of B is defined as comprising those features f such that

$$\mathrm{Sep}(m_B, f) \leq r_B + r_0 \tag{2}$$

where m_B and r_B are respectively the **midpoint** and **radius** of the translational box B^t of $B = (B^t, B^r)$ (also call them the **midpoint** and **radius** of B), and $\mathrm{Sep}(X, Y) := \inf\{\|x - y\| : x \in X, y \in Y\}$ denotes the **separation** of two Euclidean sets $X, Y \subseteq \mathbb{R}^2$. We say that B is **empty** if $\widetilde{\phi}(B)$ is empty but $\widetilde{\phi}(B_1)$ is not, where B_1 is the parent of B. We may assume the root is never empty. If B is empty, it is easy to decide whether B is `FREE` or `STUCK`: since the feature set $\widetilde{\phi}(B_1)$ is non-empty, we can find the $f_1 \in \widetilde{\phi}(B_1)$ such that $\mathrm{Sep}(m_B, f_1)$ is minimized. Then $\mathrm{Sep}(m_B, f_1) > r_B$, and by the local property of features (see Feature-Based Approach in Sec. 2), we can decide if m_B is inside (B is `STUCK`) or outside Ω (B is `FREE`).

For a box B where $B_t = S^1$, we maintain its feature set $\widetilde{\phi}(B)$ as above. But when $B_t \neq S^1$, we compute its feature set $\widetilde{\phi}(B)$ as follows. Recall that we decompose R_0 into a set of nice triangles T_j with a common apex A. For each T_j, consider the footprint of T_j with A at m_B and rotating T_j about A from θ_1 to θ_2, where $B^r = [\theta_1, \theta_2]$. By Lemma 1 the resulting swept area is a truncated triangular set (TTS); call it TTS_j. We define (cf. [15]) for a 2D shape S the **s-expansion** of S, denoted by $(S)^s$, to be the Minkowski sum of S with the $Disc(s)$ of radius s centered at the origin. For a TTS, recall that $TTS = T \cap D$ where $T = H_1 \cap H_2 \cap H_3$ is an unbounded triangular set (with each H_i a half space) and D is a disk (Figure 3). Note that $(TTS)^s$ is a proper subset of $(H_1)^s \cap (H_2)^s \cap (H_3)^s \cap (D)^s$; a theorem in the next section gives an exact representation of $(TTS)^s$. We now specify the feature

[2] The configuration space of planar rigid robots is $SE(2) = \mathbb{R}^2 \times S^1$ where S^1 is the unit circle representing angles $[0, 2\pi)$.

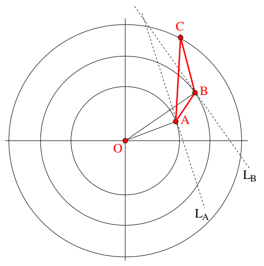


Figure 4 Nice triangle $[A, B, C]$.

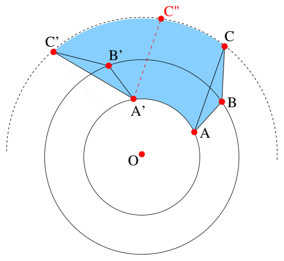


Figure 5 Nicely swept set (NSS, in blue) with A, B, C in CCW order.

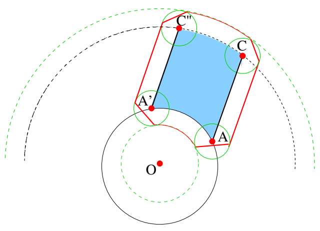


Figure 6 Expansion of $TruncStrip(A, C; A', C'')$ (in red).

set $\widetilde{\phi}(B)$: for each T_j, let $\widetilde{\phi}_j(B)$ comprise those features f satisfying $\mathrm{Sep}(m_B, f) \le r_B + r_j$ (replacing r_0 with r_j in Eq. (2)), such that f also **intersects the r_B-expansion of TTS_j**. We can think of $\widetilde{\phi}(B)$ as a collection of these $\widetilde{\phi}_j(B)$'s, each of which is used by the soft predicate $\widetilde{C}_j(B)$ so that we can apply Proposition A.

4 General Complex Robots

When R_0 is a general polygon, not necessarily star-shaped, we can still decompose R_0 into a set of triangles T_j $(j = 1, \dots, m)$, and consider the rotation of these triangles relative to a fixed point O (we may identify O with the origin). In this section, we define what it means for T_j to be "nice" relative to a point O. If O lies in the interior of T_j, we could decompose T_j into at most 6 nice pointed triangles at O, as in the previous section. Henceforth, assume that O does not lie in the interior of T_j.

4.1 Basic Representation of Nicely Swept Sets

Let $T = [A, B, C]$ be any non-degenerate triangular region defined by the vertices A, B, C. Let the origin O be outside the interior of T. We define what it means for T to be "nice relative to O". W.l.o.g., let $0 \le \|A\| \le \|B\| \le \|C\|$ where $\|A\|$ is the Euclidean norm.

We say that T is **nice** if the following three conditions hold:

$$\langle A, B - A\rangle \ge 0, \quad \langle A, C - A\rangle \ge 0, \quad \langle B, C - B\rangle \ge 0. \tag{3}$$

Here $\langle u, v\rangle$ denotes the dot product of vectors u, v.

A more geometric view of niceness is as follows (see Figure 4). Draw three concentric circles centered at O with radii $\|A\|, \|B\|, \|C\|$, respectively. Two circles would coincide if their radii are equal, but we will see that the distinctness of the vertices and niceness prevent such coincidences. Let L_A be the line tangent to the circle of radius $\|A\|$ and passing through the point A. Let H_A denote the closed half-space bounded by L_A and not containing O. The first condition in (3) $\langle A, B - A\rangle \ge 0$ says that $B \in H_A$. Similarly, the second condition says that $C \in H_A$. Finally, the last condition says that $C \in H_B$ (where H_B is analogous to H_A).

If T is a nice triangle, then $T[\alpha, \beta]$ is called a **nicely swept set (NSS)**. See Figure 5, where $T[\alpha, \beta]$ is shaded in blue. Let $T[\alpha]$ be the triangle $[A, B, C]$ and $T[\beta]$ be $[A', B', C']$. W.l.o.g., assume[3] that A, B, C appear in counter-clockwise (CCW) order as indicated in Figure 5. Then we can subdivide $T[\alpha, \beta]$ into two parts: a triangular region $[A, B, C]$ and another part which we call a **swept segment**.

[3] In case A, B, C appear in clockwise (CW) order, the boundary of $T[\alpha, \beta]$ can be similarly decomposed into two parts, comprising the swept segment $S[\alpha, \beta]$ and the triangle $[A', B', C']$.

Notation for Swept Segment: if S is the line segment $[A, C]$, then write $S[\alpha, \beta]$ for this swept segment. The boundary of $S[\alpha, \beta]$ is decomposed into the following sequence of four curves given in clockwise (CW) order: (i) the arc (A, A') centered at O of radius $\|A\|$ from A to A', (ii) the segment $[A', C']$, (iii) the arc (C', C) centered at O of radius $\|C\|$ from C' to C, (iv) the segment $[C, A]$.

Our next goal is to consider s-expansion of the swept segment, i.e.,

$$X = S[\alpha, \beta] \oplus Disc(s). \tag{4}$$

Specifically, we want an easy way to detect the intersection between this expansion with any given feature (corner or edge). To do so, we want to express X as the union of "basic shapes". A subset of $\mathbb{R}^2$ is a **0-basic shape** if it is a half-space, a disc or complement of a disc. We write $Disc(r)$ for the disc of radius r centered at O, and $Ann(r, r')$ for the annulus with inner radius r and outer radius r' centered at O. A shape X is said to be 1**-basic** if it can be written as the finite intersection $X = \bigcap_{j=1}^{k} X_j$ where X_j's are 0-basic shapes. The 1**-size** of X is the minimum k in such an intersection. So polygons with n sides have 1-size of n. Truncated triangular sets have 1-size of 4. We need some other 1-basic shapes:

- **Strips**: $Strip(a, b; a', b')$ is the region between the two parallel lines $\overline{a, b}$ and $\overline{a', b'}$. Here a, b, a', b' are distinct points.
- **Truncated strips**: $TruncStrip(a, b; a', b')$ is the intersection of $Strip(a, b; a', b')$ with an annulus; the boundary of this shape is comprised of two line segments $[a, b]$ and $[a', b']$ and two arcs (a, a') and (b, b') from the boundary of the annulus.
- **Sectors**: $Sector(a, b, b')$ denotes any region bounded by a circular arc (b, b') and two segments $[a, b]$ and $[a, b']$.

Finally, a shape X is said to be 2**-basic** if it can be written as a finite union of 1-basic shapes, $X = \bigcup_{j=1}^{m} X_j$ where X_j's are 1-basic. We call $\{X_1, \ldots, X_m\}$ a **basic representation** of X. The 2**-size** of the representation is the sum of the 1-sizes of X_j's. Thus, for any box $B_t \subseteq \mathbb{R}^2$, the s-expansion of B_t is a 2-basic shape since it is the union of four discs and an octagon. We now consider the case where X is the s-expansion of a swept segment $S[\alpha, \beta]$. We first decompose $S[\alpha, \beta]$ into two shapes as follows: suppose C'' lies on the circle of radius $\|C\| = \|C'\|$. There are two possible representations:

(1) If $[A', C'']$ is parallel to $[A, C]$ and $[A', C''] \subseteq Ann(\|A\|, \|C\|)$, then we have

$$S[\alpha, \beta] = Sector(A', C', C'') \cup TruncStrip(A, C; A', C'') \tag{5}$$

(2) If $[A, C'']$ is parallel to $[A', C']$ and $[A, C''] \subseteq Ann(\|A\|, \|C\|)$, then we have

$$S[\alpha, \beta] = Sector(A, C', C'') \cup TruncStrip(A, C''; A', C'). \tag{6}$$

The swept segment in Figure 5 supports the representation (5), but not (6). Also, if the angular range of $[\alpha, \beta]$ is greater than 90 degrees, and the points O, A, C are collinear, then both representations fail! We next show when at least one of the representations succeeds:

▶ **Lemma 4.** *Assume the width of the angular range $[\alpha, \beta]$ is at most $\pi/2$. Then swept segment $S[\alpha, \beta]$ can be decomposed into a sector and a truncated strip as in (5) or (6).*

Clearly, the s-expansion of a sector is 2-basic. This is also true for truncated strips:

▶ **Lemma 5.** *Let $X = TruncStrip(A, C; A', C'')$. There is a basic representation of $X \oplus D(s)$ of the form $\{D_1, D_2, D_3, D_4, X'\}$ where D_i's are discs and X' is the intersection of a convex hexagon with an annulus.*

Combining all these lemmas, we conclude:

▶ **Theorem 6.** *Let $T[\alpha, \beta]$ be a nicely swept set where $[\alpha, \beta]$ has width $\leq \pi/2$. Then $T[\alpha, \beta]$ can be decomposed into a triangle, a sector and a truncated strip. The s-expansion of $T[\alpha, \beta]$ has a basic representation which is the union of the s-expansions of the triangle, sector and truncated strip.*

The complexity of testing intersection of 2-basic shapes with any feature is proportional to its 2-size, which is $O(1)$. This theorem assures us that the constants in "$O(1)$" is small.

4.2 Partitioning an n-gon into Nice Triangles

Suppose P is an n-gon. We can partition it into $n-2$ triangles. W.l.o.g., there is at most one triangle that contains the origin O. We can split that triangle into at most 6 nice triangles, using our technique for star-shaped polygons (Lemma 2).

▶ **Lemma 7.** *If T is an arbitrary triangle and O is exterior to T, then we can partition T into at most 4 nice triangles.*

The number 4 in this lemma is the best possible: if T is a triangle with circumcenter O, then any partition of T into nice triangles would have at least 4 triangles because we need to introduce vertices in the middle of each side of T.

▶ **Theorem 8.** *Let P be an n-gon.*

(i) *Given any triangulation of P into $n-2$ triangles, we can refine the triangulation into a triangulation with $\leq 4n-6$ nice triangles.*

(ii) *This bound is tight in this sense: for every $n \geq 3$, there is triangulation of P whose refinement has size $4n-6$.*

4.3 Soft Predicates and T/R Subdivision Scheme

We can now follow the same paradigm as for star-shaped robots in Sec. 3.2. We first apply Theorem 8(i) to partition the robot R_0 into a set of nice triangles, $R_0 = \cup_j T_j$, where all T_j's share a common origin O, and we will use the soft predicates developed for T_j and apply Proposition A. The origin O plays a similar role as the apex in Sec. 3.2. The T/R splitting scheme is exactly the same: we first perform T-splits, splitting only the translational boxes until they are ε-small, and then we perform R-splits, splitting only the rotational boxes until they are ε-small. Essentially the top part of the subdivision tree is a quad-tree, and the bottom parts are binary subtrees (see Sec. 3.2).

The feature set for a subdivision box B where we perform T-splits is the same as before; the only difference is that now for a box B where we perform R-splits, we use a new feature set $\widetilde{\phi}_j(B)$ for each nice triangle T_j where O is not at its vertex (there are at most 6 nice triangles with O at a vertex/apex; see Theorem 8(i)). Suppose $T_j = [a, b, c]$ with $0 \leq \|a\| \leq \|b\| \leq \|c\|$. Let $r_j = \|c\|$. Also, suppose the angle range of box $B = (B^t, B^r)$ is $B^r = [\theta_1, \theta_2]$. Recall the footprint of $T_j[\theta_1, \theta_2]$ is a nicely swept set (NSS); denote it NSS_j. Then the new feature set $\widetilde{\phi}_j(B)$ for T_j comprises those f where $\mathrm{Sep}(m_B, f) \leq r_B + r_j$ and f also **intersects the r_B-expansion of** NSS_j (where m_B and r_B are the midpoint and radius of B).

5 Experimental Results

We have implemented our approaches in `C/C++` with `Qt` GUI platform. The software and data sets are freely available from the web site for our open-source `Core Library` [6]. All

Table 1 Running Our Planner (R: radius of the robot's circumcircle around its rotation center; P?: path found? (Yes/No); Time is in s; S-shaped*: thin version).

Exp#	Robot	Envir.	R	ϵ	α	β	P?	Time
0	L-shaped	gateway	50	2	(18, 98, 340°)	(458,119,270°)	Yes	10.106
1	L-shaped	gateway	50	4	(18, 98, 340°)	(458,119,270°)	No	8.431
2	snowflake	sparks	56	2	(108, 136, 0°)	(358, 155, 0°)	Yes	17.846
3	snowflake	sparks	56	2	(108, 136, 0°)	(358, 155, 180°)	Yes	3.370
4	S-shaped	sparks	74	4	(132, 80, 90°)	(333, 205, 90°)	Yes	34.284
5	S-shaped	sparks	74	4	(132, 80, 90°)	(333, 205, 60°)	No	57.371
6	3-legged	sparks	70	2	(108, 136, 0°)	(368, 155, 0°)	Yes	41.745
7	L-shaped	corridor	68	2	(75, 420, 0°)	(370, 420, 0°)	Yes	4.012
8	L-shaped	corridor	68	3	(75, 420, 0°)	(370, 420, 0°)	Yes	1.926
9	L-shaped	corridor	68	5	(75, 420, 0°)	(370, 420, 0°)	Yes	2.684
10	L-shaped	corridor-L	68	5	(75, 420, 0°)	(370, 420, 0°)	No	2.908
11	L-shaped	corridor-L	68	3	(75, 420, 0°)	(370, 420, 0°)	Yes	2.255
12	C-shaped	corridor-S	80	4	(80, 450, 0°)	(380, 450, 0°)	Yes	26.200
13	S-shaped	maze	38	2	(38, 38, 0°)	(474, 474, 90°)	No	90.097
14	S-shaped*	maze	38	2	(38, 38, 0°)	(474, 474, 90°)	Yes	79.518

Table 2 Comparing with OMPL ("#": Exp#; "Time/P?": our run time (in s)/path found? (Y/N). Each OMPL method: Average Time (in s)/Standard Deviation/Success Rate, over 10 runs).

#	Time/P?	PRM	RRT	EST	KPIECE
0	10.106/Y	**4.18**/2.53/1	42.13/38.49/1	76.22/110.44/0.9	300/0/0
2	17.846/Y	**9.22**/6.82/1	210.41/144.25/0.3	271.75/89.31/0.1	240.00/126.47/0.2
3	**3.370**/Y	300/0/0	300/0/0	300/0/0	300/0/0
4	34.284/Y	**5.93**/7.20/1	217.33/134.53/0.3	300/0/0	300/0/0
5	**57.371**/N	300/0/0	300/0/0	300/0/0	300/0/0
6	41.745/Y	**2.72**/4.89/1	154.22/141.77/0.5	104.32/78.10/0.7	3.16/4.28/1
8	1.926/Y	0.63/0.55/1	300/0/0	3.02/4.71/1	**0.41**/0.28/1
11	2.255/Y	**1.49**/0.84/1	300/0/0	241.24/124.88/0.2	1.58/1.47/1
12	26.200/Y	**3.16**/4.21/1	300/0/0	172.506/120.38/0.7	93.88/88.03/0.8
13	**90.097**/N	300/0/0	300/0/0	300/0/0	300/0/0
14	79.518/Y	300/0/0	236.72/106.44/0.3	300/0/0	**39.81**/91.57/0.9

experiments are reproducible as targets of Makefiles in `Core Library`. Our experiments are on a PC with one 3.4GHz Intel Quad Core i7-2600 CPU, 16GB RAM, nVidia GeForce GTX 570 graphics and Linux Ubuntu 16.04 OS. The results are summarized in Table 1 and Table 2. Table 1 is only concerned with the behavior of our complex robots; Table 2 gives comparisons with the open-source OMPL library [14]. The robots are as shown in Figure 1.

We select some interesting experiments to explain characteristic behavior of our planner. Please see Table 1 and the video (`https://cs.nyu.edu/exact/gallery/complex/complex-robot-demo.mp4`). In Exp0-1, we show how the parameter ϵ affects the result. With a narrow gateway, when we change ϵ from 2 to 4, the output changes from a path to `NO-PATH` for the same configuration. In Exp2-3, we observe how the snowflake robot rotates and maneuvers to get from the start to two different goals. For Exp4-5, the difference is in the angles of the goal configuration; in Exp5 this is designed to be an isolated configuration

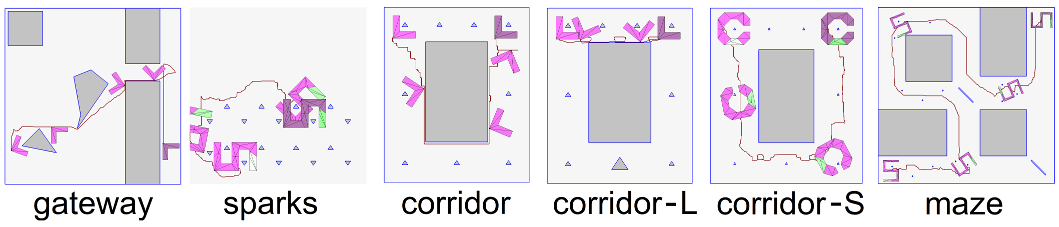


Figure 7 Six Environments in our experiments.

and the planner outputs NO-PATH as desired. Exp6 shows how the robot can move to use its complex shape and the environment to squeeze through the obstacles. Exp7-9 are of the same configuration with only the differences in ϵ. The planner can find three totally different paths. When ϵ is small (Exp7), the path is very carefully adjusted to move the robot around the obstacles. When ϵ is larger (Exp8), the planner finds an upper path with a higher clearance. When ϵ is even larger (Exp9), the planner chooses a very safe but much longer path at the bottom. Note that using a larger ϵ usually makes the search faster, since we stop splitting boxes smaller than ϵ, but a longer path can make the search slower. In Exp10-11, we modify the environment of Exp7-9 by putting a large obstacle at the bottom, which forces the robot to find a path at the top. Exp12 uses an environment similar to those in Exp7-11 but with much smaller scattered obstacles. It is designed for the C-shaped robot, which can rotate while having an obstacle in its pocket. Exp13-14 use a challenging environment where the small scattered obstacles force the S-shaped robot to rotate around and only the "thin" version (Exp14, also in Fig. 7 "maze") can squeeze through.

In Table 2 we compare our planner with several sampling algorithms in OMPL: PRM, RRT, EST, and KPIECE. These experiments are correlated to those in Table 1 (see the Exp #). Each OMPL planner is run 10 times with a time limit 300 seconds (default), where all planner-specific parameters use the OMPL default values. We see that for OMPL planners there are often unsuccessful runs and they have to time out even when there is a path. On the other hand, our algorithm consistently solves the problems in a reasonable amount of time, often much faster than the OMPL planners, in addition to being able to report NO-PATH.

6 Conclusions

Although the study of rigorous algorithms for motion planning has been around for over 40 years, there has always been a gap between such theoretical algorithms and the practical methods. Our introduction of resolution-exactness and soft predicates on the theoretical front, together with matching implementations, closes this gap. Moreover, it eliminated the "narrow passage" problem that plagued the sampling approaches. The present paper extends our approach to challenging planning problems for which no exact algorithms exist.

What are the current limitations of our work? We implement everything in machine precision (the practice in this field). But it can be easily modified to achieve the theoretical guarantees of resolution-exactness if we use arbitrary precision BigFloats number types.

We pose two open problems: One is to find an optimal decomposition of m-gons into nice triangles (currently, we simply give an upper bound). Such decompositions will have impact for practical complex robots. Second, we would like to develop similar decomposability of soft predicates for complex rigid robots in $\mathbb{R}^3$.

References

1 F. Avnaim, J-D. Boissonnat, and B. Faverjon. A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles. In Boissonnat J.D. and Laumond J.P., editors, *Geometry and Robotics*, LNCS Vol 391. Springer, Berlin-Heidelberg, 1989.

2 M. Barbehenn and S. Hutchinson. Efficient search and hierarchical motion planning by dynamically maintaining single-source shortest paths trees. *IEEE Trans. Robotics and Automation*, 11(2), 1995.

3 J. Basch, L.J. Guibas, D. Hsu, and A. Nguyen. Disconnection proofs for motion planning. In *IEEE Int'l Conf. on Robotics Animation*, pages 1765–1772, 2001.

4 Huxley Bennett, Evanthia Papadopoulou, and Chee Yap. Planar minimization diagrams via subdivision with applications to anisotropic Voronoi diagrams. *Eurographics Symposium on Geometric Processing*, 35(5), 2016. SGP 2016, Berlin, Germany. June 20-24, 2016.

5 H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations.* MIT Press, Boston, 2005.

6 Core Library. `https://cs.nyu.edu/exact/core_pages/downloads.html`.

7 Dan Halperin, Efi Fogel, and Ron Wein. *CGAL Arrangements and Their Applications.* Springer-Verlag, Berlin and Heidelberg, 2012.

8 Ching-Hsiang Hsu, Yi-Jen Chiang, and Chee Yap. Rods and rings: Soft subdivision planner for **R**^3 x **S**^2, 2018. Available at `http://cse.poly.edu/chiang/rod-ring18.pdf`.

9 Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

10 Steven M. LaValle. *Planning Algorithms.* Cambridge University Press, Cambridge, 2006.

11 Zhongdi Luo, Yi-Jen Chiang, Jyh-Ming Lien, and Chee Yap. Resolution exact algorithms for link robots. In *Proc. 11th Intl. Workshop on Algorithmic Foundations of Robotics (WAFR '14)*, volume 107 of *Springer Tracts in Advanced Robotics (STAR)*, pages 353–370, 2015. 3-5 Aug 2014, Boğazici University, Istanbul, Turkey.

12 Victor Milenkovic, Elisha Sacks, and Steven Trac. Robust complete path planning in the plane. In *Proc. 10th Workshop on Algorithmic Foundations of Robotics (WAFR 2012)*, Springer Tracts in Advanced Robotics, vol.86, pages 37–52. Springer, 2012.

13 O. Salzman, M. Hemmer, B. Raveh, and D. Halperin. Motion planning via manifold samples. In *Proc. European Symp. Algorithms (ESA)*, 2011.

14 I.A. Şucan, M. Moll, and L.E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012. `doi:10.1109/MRA.2012.2205651`.

15 Cong Wang, Yi-Jen Chiang, and Chee Yap. On Soft Predicates in Subdivision Motion Planning. *Comput. Geometry: Theory and Appl. (Special Issue for SoCG'13)*, 48(8):589–605, 2015.

16 Chee Yap, Zhongdi Luo, and Ching-Hsiang Hsu. Resolution-exact planner for thick non-crossing 2-link robots. In *Proc. 12th Intl. Workshop on Algorithmic Foundations of Robotics (WAFR '16)*, 2016. 13-16 Dec 2016, San Francisco. The appendix in the full paper (and arXiv from `http://cs.nyu.edu/exact/` (and `arXiv:1704.05123 [cs.CG]`) contains proofs and additional experimental data.

17 Chee K. Yap. Soft Subdivision Search in Motion Planning. In A. Aladren et al., editor, *Proceedings, 1st Workshop on Robotics Challenge and Vision (RCV 2013)*, 2013. Robotics Science and Systems Conference (RSS 2013), Berlin. In arXiv:1402.3213. Full paper: http://cs.nyu.edu/exact/papers/.

18 Chee K. Yap. Soft Subdivision Search and Motion Planning, II: Axiomatics. In *Frontiers in Algorithmics*, volume 9130 of *Lecture Notes in Comp.Sci.*, pages 7–22. Springer, 2015. Plenary Talk at 9th FAW. Guilin, China. Aug 3-5, 2015.

19 Liangjun Zhang, Young J. Kim, and Dinesh Manocha. Efficient cell labeling and path non-existence computation using C-obstacle query. *Int'l. J. Robotics Research*, 27(11–12):1246–1257, 2008.

20 Bo Zhou, Yi-Jen Chiang, and Chee Yap. Soft subdivision motion planning for complex planar robots, 2018. Full version available at `http://cse.poly.edu/chiang/esa18-full.pdf`.

21 D.J. Zhu and J.-C. Latombe. New heuristic algorithms for efficient hierarchical path planning. *IEEE Transactions on Robotics and Automation*, 7:9–20, 1991.